Real-Time Systems, 6, 173–206 (1994)

Compositional Verification of a Distributed Real-Time Arbitration Protocol

JOZEF HOOMAN wsinjh@win.tue.nl
Department of Mathematics and Computing Science, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands

Abstract. A distributed real-time arbitration protocol is specified and verified using an assertional method. The formalism is based on classical Hoare triples which have been extended to deal with real-time properties. To verify design steps, a compositional proof system has been formulated for these extended triples. The intention of the protocol is to resolve contention between a number of concurrent modules that compete to acquire control of a common bus. Therefore our proof method has been adapted to deal with concurrent processes that communicate by means of a common bus. Compositionality makes it possible to verify the required properties of the protocol using only the specifications of the modules. Next we give a top-down derivation of a program implementing a module according to its real-time specification.

1. Introduction

During the past decade a large number of formal methods have been devised to specify and verify real-time systems, see for instance [2]. The main aim of this paper is not to introduce yet another formalism, but to investigate the application of an existing formal framework to a small but realistic example. The formalism is only adapted as far as it is required by this example. We consider a distributed real-time arbitration protocol in which a number of distributed components asynchronously execute a certain algorithm. A well-known verification problem is that the number of possible states of such a system is exponential in the number of components. To deal with this state explosion problem we have chosen to use an assertional method in which a set of states can be characterized by a single logical formula. Further, to reduce the complexity of the verification task, we use a method which is compositional, that is, which allows reasoning with the specifications of components rather than their implementation.

Observing the nice compositional rules for sequential composition and iteration of classical Hoare triples (precondition, program, postcondition), we have modified these triples to obtain a formalism for the specification and the verification of distributed real-time systems. The assertion language, in which pre- and postcondition are expressed, has been extended with primitives denoting the timing of actions. Since in classical Hoare triples the postcondition only has to hold for terminating computations, they are restricted to the verification of safety properties. Our aim is to specify and verify progress properties of both terminating and nonterminating computations. Therefore we have adapted the interpretation of triples. To indicate that we deviate from classical triples $\{p\}\ S\ \{q\}$, by extending the assertion language and the interpretation, we use the notation $\langle\langle p\rangle\rangle\ S\ \langle\langle q\rangle\rangle$, where p and q are assertions and S is a process. Further, p is called the *assumption* and q the *commitment* (instead of, respectively., precondition and postcondition). The main idea is that if the assumption holds for any computation of the program, then it also satisfies the commitment.

To verify the correctness of a program with respect to an assumption/commitment pair, we give a proof system, that is, a set of rules and axioms by which valid triples can be derived. To allow the verification of design steps during the process of top-down program development, we formulate a proof system which is compositional. This means that the specification of each compound programming language construct (such as sequential composition and parallel composition) can be deduced from specifications for its constituent parts without any further information about the internal structure of these parts. Compositionality can be considered as a prerequisite for hierarchical, structured, program derivation.

In previous work, e.g. [5], a preliminary version of our Hoare-style framework has been used to specify and verify distributed real-time systems in which parallel processes communicate by synchronous message passing along channels. We have axiomatized the maximal parallelism model, where each process has its own processor, as well as multiprogramming with several processes on a single processor and scheduling based on priorities. In [7] the formalism described here has been extended to deal with asynchronous communication along channels and it has been applied to a process control example of a water level monitoring system.

In this paper we investigate the flexibility of our approach by considering a different communication mechanism where processes communicate by means of a common bus. At any point in time at most one process, also called module, can transfer data along the bus. We study a protocol which is intended to resolve contention when two or more modules want to perform data transactions. To perform arbitration between competing modules, the bus has a number of special lines which are used to start the protocol and to determine which module gets access to the data lines of the bus. Here we concentrate on the arbitration lines, and consider the situation as depicted in figure 1. The bus consists of $k+1$ arbitration lines $bus[0], \ldots, bus[k]$ (the data lines are not represented in the picture). There are $n+1$ processes $MOD_0, \ldots, MOD_n$, that are connected via, respectively, links $link_0, \ldots, link_n$ to the arbitration lines of the bus. Similar to the bus, we assume that each link consists of $k+1$ lines. The modules can apply values to the lines of their links. A link line carries the last applied value, and the value of a bus line is a combination of the values carried by all corresponding link lines. Further, modules can read the values of the bus lines.

We consider a distributed real-time arbitration protocol which is inspired by an algorithm of the IEEE 896 Futurebus specification [8]. The original specification is hardware oriented and all values are 0 or 1. The value of bus line j, i.e., $bus[j]$, is then obtained by taking the OR-function of the values carried by $link_0[j], \ldots, link_n[j]$, for $j = 0, \ldots, k$. Thus a bus line carries the value 0 iff all cor-

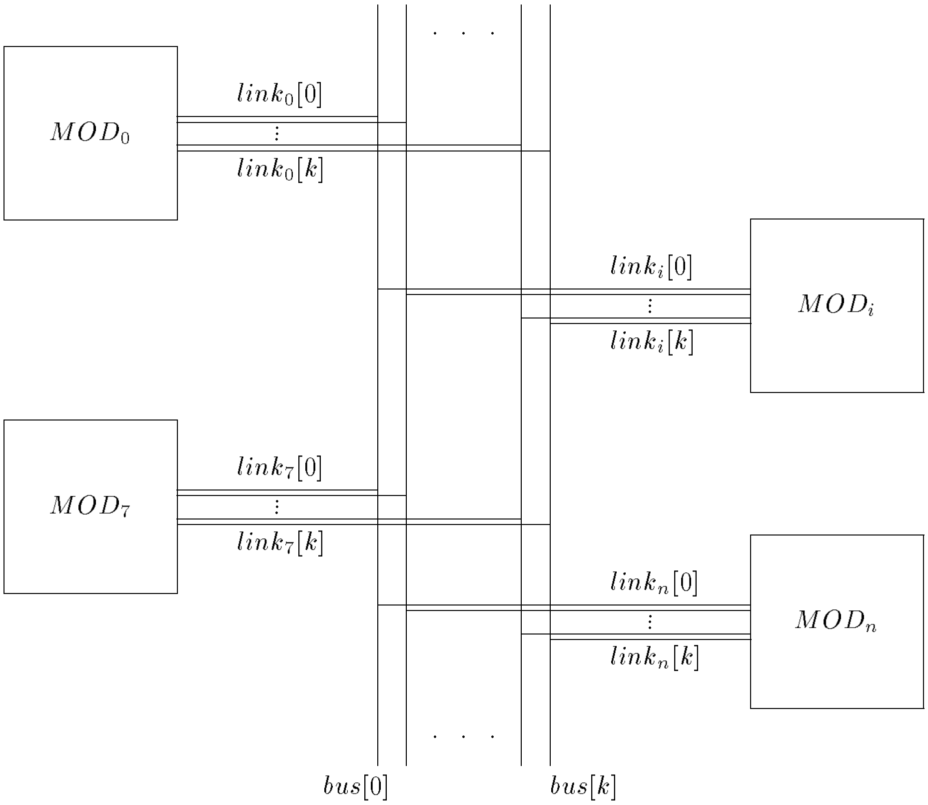

Figure 1. Modules communicating via a common bus.

responding link lines carry 0. The protocol is used to resolve contention between modules that are competing to acquire control of the bus in order to perform data transactions. Each module MOD_i has a unique arbitration number represented by $k+1$ bits, $b_0^i, \ldots, b_k^i$, where b_0^i is the most significant bit. Initially, all bus lines carry the value 0.

All modules, asynchronously, execute the following algorithm. First they apply their own arbitration number to their link. Next module MOD_i, for $i = 0, \ldots, n$, periodically reads the bus lines and applies b_j^i to $link_i[j]$ if it has not detected a bus value greater than its own bit on the more significant positions $0, \ldots, j-1$. Otherwise 0 is applied to the link line. Thus if MOD_i detects that the value of $bus[j]$ is greater than b_j^i then it applies 0 to the lines $link_i[j+1], \ldots, link_i[k]$. The modules need not start their algorithm at exactly the same point in time and they may execute it with different speed. The aim is to show that after a fixed period of time the value of the bus will be stable, showing the number of the winner. Further, we have to compute the maximum time it takes to stabilize, because the modules should terminate the algorithm after stabilization and read the bus to determine the winner. Hence the modules should have a constant representing this maximum duration.

Example: Consider three competing modules MOD_0, MOD_1, and MOD_2 with arbitration numbers 1101, 1011, and 0111, respectively. When these modules have applied their numbers to their links, the bus carries the value 1111. Next, after reading this value, MOD_0, MOD_1, and MOD_2 will apply, respectively, 1100, 1000, and 0000. This leads to 1100 on the bus. Given this value of the bus they apply, respectively, 1101, 1000, and 0000, which yields bus value 1101, i.e. the number of the winner (MOD_0), and a stable situation. Note that this only describes a very simple execution. The bus might have several other intermediate values, since the modules are not synchronized and may execute with different speed. □

To investigate the essential properties of the protocol, we generalize it and use nonnegative values instead of bits. Then the value of a bus line is the maximum of the values carried by the corresponding link lines. Further, we allow some delay between a change of a link and a corresponding change of the value of the bus. In the proof system we axiomatize the properties of the bus and formulate rules for the read and the apply actions. Then we apply the resulting proof method to the generalized arbitration protocol.

Using compositionality of our proof system, we first abstract from the implementation of modules. Only their specifications are used to prove the required property (stabilization after a certain period of time). At this level of the verification, when dealing with a number of parallel processes, we mainly reason by means of the commitments of our extended Hoare triples. Next we derive an implementation of a module in a top-down way, starting from its real-time specification. Similar to the non-real-time case, it turns out that the structure of our triples, with assumptions and commitments, is very convenient for the derivation of sequential programs.

To compare different approaches, we have investigated in [5] an assertional method based on a real-time version of temporal logic, called Metric Temporal Logic [9]. An alternative approach can be found in [11, 4] where Explicit Clock Temporal Logic has been used, extending the non-compositional method of [10] to real-time. Close to our compositional approach is the top-down design method of the Duration Calculus [1], a real-time interval logic based on the concept of the duration of states. Related to our formal framework is an early paper by Haase [3] in which real-time is introduced as a variable in the data space of the program and assertions are derived by Dijkstra's weakest precondition calculus. Recently, in [12] a non-compositional method based on proof outlines with control predicates has been extended to reason about timing properties of concurrent programs that communicate via shared variables.

This paper is structured as follows. In chapter 2 the formal framework is introduced as far as it is needed in chapter 3, where we verify the required stabilization property using specifications of the modules. To implement these modules, we describe in chapter 4 a real-time concurrent programming language with communication by means of a common bus. For this language a compositional axiomatization is formulated in chapter 5. By means of this proof system we derive in chapter 6 an implementation of a module according to its specification. Concluding remarks can be found in chapter 7. A preliminary version of this paper, using a predecessor of the formalism, appeared in [6].

2. Formal Framework

We introduce a formal method to specify and verify distributed real-time systems. This formalism will be applied to the arbitration protocol. In this chapter we only consider the part of the formalism which is needed to formalize the first stage of the verification, showing that the algorithm performed by the modules, as expressed by their specifications, leads to the required arbitration properties.

As already mentioned in the introduction, our assertional method is based on triples of the form $\langle\langle p\rangle\rangle\ S\ \langle\langle q\rangle\rangle$, where S is a program, and p and q are assertions called, respectively, *assumption* and *commitment*. In the next section we discuss the parallel composition operator of the programming language. The assertion language is described in section 2.2 where we also axiomatize the properties of the common bus. A number of small examples can be found in section 2.3. Finally, in section 2.4, we mention the proof rules that will be used in the next chapter.

2.1. *Parallel Composition of Processes*

Let VAR be a nonempty set of program variables, ranging over a valued domain VAL. Let $LINK = \{link_0, \ldots, link_n\}$ be a finite set of links to the bus. Link $link_i$ consists of $k+1$ lines $link_i[0], \ldots, link_i[k]$, for $i = 1, \ldots, n$.

During the first phase of the verification we consider parallel components that can read the value of a common bus and can apply a value to a link. No information is needed about the implementation of the modules. Therefore in this section only the parallel composition operator is mentioned.

- $S_1 \| S_2$ indicates parallel execution of the components S_1 and S_2. We require that S_1 and S_2 do not share links and do not have shared variables. Thus parallel components may only communicate by means of the common bus. The components of a parallel composition are often called *processes* or *modules*.

Let $var(S)$ be the set of program variables that might be changed by process S. Similarly, $lnk(S)$ is the set of links on which S might apply values. Thus $var(S) \subseteq VAR$ and $lnk(S) \subseteq LINK$. Hence for $S_1 \| S_2$ we have $var(S_1) \cap var(S_2) = \emptyset$ and $lnk(S_1) \cap lnk(S_2) = \emptyset$.

2.1.1. *Timing assumptions*

Clearly, the timed behaviour of a parallel program depends on the number of available processors. Thus, to prove real-time properties about networks of processes, we have to make certain assumptions about the underlying execution mechanism of parallelism. In this paper we assume maximal parallelism, representing the situation that each parallel process has its own processor. Further, we use parameters to express the delay between a change of the value of a link line (caused by an apply action) and its effect on the value of the corresponding bus line.

- For bus line j we have a parameter TB_j such that the value of a link line contributes to the value of bus line j after TB_j time units, for $j = 0, \ldots, k$.

2.2. *Assertions*

We express the timing behaviour of an embedded real-time system from the viewpoint of an external observer with his own clock. Thus, although parallel components of a system might have their own local clock, the observable behaviour of a system is described in terms of a single, conceptual, global clock. Since this global notion of time is not incorporated in the distributed system itself, it does not impose any synchronization upon processes. In this paper we use a time domain *TIME* which is dense, i.e., between every two points of time there exists an intermediate point. Having dense time is convenient to describe systems in which actions may occur arbitrarily close to each other in time. This is for instance the case in the arbitration example where concurrent modules need not start at the same time and the execution speed of different modules need not be related. We assume that $\mathbb{N} \subseteq \mathit{VAL} \subseteq \mathit{TIME}$, where $\mathbb{N}$ denotes the set of natural numbers (including 0).
In a triple $\langle\langle p \rangle\rangle \; S \; \langle\langle q \rangle\rangle$, assertion p expresses assumptions about

- the initial state of S, i.e., the values of the program variables at the start of the execution of S,
- the starting time of S, and
- the timed occurrence of observable events.

Given assumption p, assertion q expresses a commitment of S,

- if S terminates, about the final state of S, i.e., the values of the program variables at termination,
- about the termination time (∞ if S does not terminate), and
- about the timed occurrences of observable events.

Note that, in contrast with classical Hoare triples, our commitment expresses properties of terminating as well as nonterminating computations. Together with the addition of time, this means that our formalism is not restricted to partial correctness or safety properties, but allows the specification and verification of progress properties.

The assertions p and q in a correctness formula $\langle\langle p\rangle\rangle\ S\ \langle\langle q\rangle\rangle$ are expressed in a first-order logic with the following primitives.

- Program variables, such as $x, y \in VAR$, ranging over VAL.
- Logical value variables, such as v, v_0, v_1, ..., ranging over VAL, and logical time variables, such as t, t_0, t_1, ..., ranging over $TIME \cup \{\infty\}$.
- A special variable $time$, ranging over $TIME \cup \{\infty\}$, which refers to our global notion of time. An occurrence of $time$ in assumption p represents the starting time of the statement S whereas in commitment q it denotes the termination time (using $time = \infty$ for nonterminating computations).

To reason about natural numbers, the assertion language includes the predicate

- $exp \in I\!N$ which is true iff the value of expression exp is a natural number.

Henceforth we use variables i, j, l, m that range over $I\!N$. Thus, for instance, $\forall j : p$ is an abbreviation of $\forall j, j \in I\!N : p$.

To reason about modules that communicate by means of a common bus, we assume that the apply actions and the value of the bus are observable at any point in time. In the sequel we use exp to represent an expression which yields a value in the time domain $TIME$, whereas e is an expression ranging over VAL. We have

- $(apply, link[j], e)$ **at** exp, for $j = 0, \ldots, k$, to denote that the value of e is applied to line j of the link $link$ at time exp.
- $bus[j]$ **at** exp, for $j = 0, \ldots, k$, to denote the value of bus line j at time exp.

Note that $(apply, link[j], e)$ **at** exp is a predicate, which is true or false, and that $bus[j]$ **at** exp yields a value in VAL.

Let $var(p)$ be the set of program variables occurring in assertion p. Similarly, $lnk(p)$ is the set of links occurring in p. Further $p[exp/var]$ denotes the substitution of expression exp for each free occurrence of variable var in assertion p.

We often use intervals of time points such as $[t_0, t_1\rangle = \{t \in TIME \mid t_0 \leq t < t_1\}$, $\langle t_0, t_1\rangle = \{t \in TIME \mid t_0 < t < t_1\}$, etc. Furthermore, we define the following abbreviations.

- For a set $I \subseteq TIME$ (usually an interval) and a predicate P **at** exp,
  $P\ \mathbf{at}\ I \equiv \forall t \in I : P\ \mathbf{at}\ t$
  $P\ \mathbf{in}\ I \equiv \exists t \in I : P\ \mathbf{at}\ t$
- $(apply, link[j])\ \mathbf{at}\ exp\ \equiv\ \exists v : (apply, link[j], v)\ \mathbf{at}\ exp$
  $\neg(apply, link[j])\ \mathbf{at}\ exp\ \equiv\ \neg((apply, link[j])\ \mathbf{at}\ exp)$
  $\neg(apply, link)\ \mathbf{at}\ exp\ \equiv\ \forall j \leq k : \neg(apply, link[j])\ \mathbf{at}\ exp$

- Define $link[j]$ **at** exp, that is the value of line $link[j]$ at time exp, as follows.
$$link[j] \textbf{ at } exp = \begin{cases} 0 & \text{if } \neg(apply, link[j]) \textbf{ at } [0, exp] \\ e & \text{if } \exists t_0 \leq exp : (apply, link[j], e) \textbf{ at } t_0 \wedge \\ & \qquad\qquad \neg(apply, link[j]) \textbf{ at } \langle t_0, exp] \end{cases}$$

- Instead of $(bus[j] \textbf{ at } exp) = e$ we also use $(bus[j] = e) \textbf{ at } exp$. Similarly, with $=$ replaced by $>$, $\geq$, $<$, etc., and $bus[j]$ replaced by $link[j]$.

The following lemma follows directly from the definitions.

Lemma 2.1 For a predicate P **at** exp we have $\neg(P \textbf{ at } I) \leftrightarrow (\neg P) \textbf{ in } I$ and $(\neg P) \textbf{ at } I \leftrightarrow \neg(P \textbf{ in } I)$.

The next lemma expresses that if we apply the value v to a link in the interval $[t_1, t_2\rangle$ and no other values are applied to the link in $[t_1, t_3\rangle$, then the link has value v during $[t_2, t_3\rangle$.

Lemma 2.2 If $(apply, link[j], v) \textbf{ in } [t_1, t_2\rangle$ and, for all $t \in [t_1, t_3\rangle$, the assertion $(apply, link[j], v_0) \textbf{ at } t$ implies $v_0 = v$, then $(link[j] = v) \textbf{ at } [t_2, t_3\rangle$.

Bus Property

We axiomatize the values of the bus lines, i.e., we express that the value of bus line j is, after some delay TB_j, the maximum of the values of the corresponding links. Recall that $LINK = \{link_0, \ldots, link_n\}$, that is, only the links $link_0, \ldots, link_n$ affect the value of the bus. This leads to the following property:

$$\boxed{\forall j \leq k \, \forall t \; : \; bus[j] \textbf{ at } (t + TB_j) \; = \; max(\, link_0[j] \textbf{ at } t \,, \; \ldots \;, \; link_n[j] \textbf{ at } t \,)}$$

2.3. Examples

We give a few small examples to illustrate our real-time version of Hoare triples. In the first triple we consider an assignment $x := x + 7$ which starts at time 6 in a state where x has the value 5 and assuming that the value 0 has been applied to line 2 of $link_4$ at time 3. Suppose the execution of the assignment takes T_a time units, for some parameter T_a. Then the commitment expresses that this assignment terminates at time $6 + T_a$ in a state where x has the value 12. Further, given the assumption, we can also assert in the commitment that the value 0 has been applied to line 2 of $link_4$ at 3.

$$\begin{array}{l} \langle\!\langle x = 5 \wedge time = 6 \wedge (apply, link_4[2], 0) \textbf{ at } 3 \rangle\!\rangle \\ \qquad x := x + 7 \\ \langle\!\langle x = 12 \wedge time = 6 + T_a \wedge (apply, link_4[2], 0) \textbf{ at } 3 \rangle\!\rangle. \end{array}$$

Specifications can be made more general by using logical variables to represent the initial value of program variables and the starting time. For instance,

$$\langle\!\langle x = v \wedge time = t < \infty \rangle\!\rangle \; FUN \; \langle\!\langle y = f(v) \wedge x = v \wedge t + 5 < time < t + 13 \rangle\!\rangle.$$

This triple specifies that program FUN should compute $f(x)$ within certain time bounds, for a particular function f, leaving x invariant. Logical variables such as v and t are implicitly universally quantified. Also nonterminating computations can be specified. For instance, for some parameter T, we can specify that program $LOOP$ does not terminate and applies value i to all lines of a link $link$ at time $i \times T$, for all $i \in I\!N$.

$$\begin{array}{l} \langle\!\langle x = 0 \wedge time = 0 \rangle\!\rangle \\ \qquad LOOP \\ \langle\!\langle time = \infty \wedge \forall i \, \forall j \leq k : (apply, link[j], i) \textbf{ at } (i \times T) \rangle\!\rangle. \end{array}$$

Further a program might have both terminating and nonterminating computations:

$$\begin{array}{l} \langle\!\langle x = v \in I\!N \wedge time = 0 \rangle\!\rangle \\ \qquad ITERATE \\ \langle\!\langle (v < 25 \wedge time = \infty) \vee (v \geq 25 \wedge time = (v - 25) \times T \wedge x = 0) \rangle\!\rangle. \end{array}$$

Note that a classical Hoare triple $\{p\} \; S \; \{q\}$, denoting partial correctness, can be expressed in our framework by

$$\langle\!\langle p \wedge time < \infty \rangle\!\rangle \; S \; \langle\!\langle time < \infty \rightarrow q \rangle\!\rangle.$$

Similarly, total correctness of S with respect to p and q can be expressed as

$$\langle\!\langle p \wedge time < \infty \rangle\!\rangle \; S \; \langle\!\langle time < \infty \wedge q \rangle\!\rangle.$$

2.4. *Proof Rules*

In this section we only consider the proof rules which are needed for the first verification step of the arbitration protocol. More rules and axioms will be given in section 5.2, to derive an implementation of the modules. Here we start with the consequence rule which is identical to the classical rules for Hoare triples.

Rule 2.1 (Consequence)
$$\frac{\langle\!\langle p_0 \rangle\!\rangle \; S \; \langle\!\langle q_0 \rangle\!\rangle, p \rightarrow p_0, q_0 \rightarrow q}{\langle\!\langle p \rangle\!\rangle \; S \; \langle\!\langle q \rangle\!\rangle}$$

Further the proof system contains rules for parallel composition. If the special variable $time$ does not occur in the commitments of the components of a parallel composition, then we have the following simple rule.

Rule 2.2 (Simple Parallel Composition)

$$\frac{\langle\!\langle p_1 \rangle\!\rangle \; S_1 \; \langle\!\langle q_1 \rangle\!\rangle, \qquad \langle\!\langle p_2 \rangle\!\rangle \; S_2 \; \langle\!\langle q_2 \rangle\!\rangle}{\langle\!\langle p_1 \wedge p_2 \rangle\!\rangle \; S_1 \| S_2 \; \langle\!\langle q_1 \wedge q_2 \rangle\!\rangle}$$

provided, for $i = 1, 2$, $var(q_i) \subseteq var(S_i)$, $lnk(p_i, q_i) \subseteq lnk(S_i)$, and the special variable $time$ does not occur in q_i.

Note that the rule has several restrictions to ensure its soundness. The first condition requires that the commitment of a process only refers to program variables of the process itself. By the second restriction the specification of a process should only refer to its own interface, i.e., its own links. The problem with $time$ in the commitments is that, in general, the termination times of S_1 and S_2 will be different. To formulate a more general rule for parallel composition, we substitute a logical variable t_i for $time$ in q_i, $i = 1, 2$. Then the termination time of $S_1 \| S_2$, expressed by $time$ in its commitment, is the maximum of t_1 and t_2. Further, for completeness, we add predicates to express that process S_i does not perform any apply action on its links between t_i and $time$. Define, for a set of links $lset \subseteq LINK$,

$$(\neg lset) \textbf{ at } t \equiv \bigwedge_{link \in lset} \neg(apply, link) \textbf{ at } t.$$

(By convention, $(\neg\emptyset)$ **at** $t \leftrightarrow true$.) Then we have the following rule, where t_1 and t_2 are fresh logical variables.

Rule 2.3 (Parallel Composition)

$$\frac{\begin{array}{l} \langle\langle p_1 \rangle\rangle \; S_1 \; \langle\langle q_1 \rangle\rangle, \qquad \langle\langle p_2 \rangle\rangle \; S_2 \; \langle\langle q_2 \rangle\rangle \\ q_1[t_1/time] \wedge (\neg lnk(S_1)) \textbf{ at } [t_1, time\rangle \wedge \\ q_2[t_2/time] \wedge (\neg lnk(S_2)) \textbf{ at } [t_2, time\rangle \wedge time = max(t_1, t_2) \rightarrow q \end{array}}{\langle\langle p_1 \wedge p_2 \rangle\rangle \; S_1 \| S_2 \; \langle\langle q \rangle\rangle}$$

provided, for $i = 1, 2$, $var(q_i) \subseteq var(S_i)$ *and* $lnk(p_i, q_i) \subseteq lnk(S_i)$.

Note that this is a compositional rule, since a specification of the compound construct $S_1 \| S_2$ can be derived without information about the program text of the components S_1 and S_2, using only their specifications and the functions var and lnk which specify the static interface of these processes.

3. Example Distributed Real-Time Arbitration Protocol

As an example of our compositional approach, we specify and verify the distributed real-time arbitration protocol introduced in chapter 1. Our algorithm is based on a protocol occurring in the IEEE 896 Futurebus specification [8] which has been generalized by allowing arbitrary nonnegative values instead of bits. In this chapter we formulate an assertional specification of the modules and prove that these specifications lead to the required properties.

Suppose there are $n + 1$ modules, $MOD_0, \ldots, MOD_n$, that are competing to acquire control of the bus in order to carry out data transactions (see figure 1 in chapter 1). To resolve contention, each module MOD_i, for $i = 0, \ldots, n$, has a unique arbitration number which is, for a bus of $k + 1$ lines, represented by $k + 1$ values $b^i_0, \ldots, b^i_k$. Let b^i be the list of values $b^i_0, \ldots, b^i_k$ and $val(b^i)$ be the value represented by b^i. We assume that values are nonnegative and that b^i_0 is the most significant value, i.e.,

(A1) $b^i_j \geq 0$, for $i = 0, \ldots, n$, $j = 0, \ldots, k$.

(A2) $val(b^{i_1}) > val(b^{i_2}) \rightarrow \exists l \leq k : b_l^{i_1} > b_l^{i_2} \wedge \forall j < l : b_j^{i_1} = b_j^{i_2}$,
for $i_1, i_2 \in \{0, \ldots, n\}$.

When two or more modules compete for the bus, the winner will be the one with the highest arbitration number. The idea is that after a suitable setting time T the bus will carry the highest arbitration number of the modules competing, thus indicating the winner of the arbitration. This is achieved by an algorithm in which each module MOD_i, for $i = 0, \ldots, n$, reads the bus and applies values to its link $link_i$. Henceforth we only describe modules that compete for the bus, assuming that modules that are not interested to acquire control of the bus always apply the value 0 to their link. The competing modules need not start at the same time, but we assume that they start before some time δ_0^i. Further, after time T all modules terminate the arbitration algorithm and they can inspect the bus to determine which module acquires the bus. We give an abstract description of the protocol by means of six commitments that should hold for MOD_i, for all $t < time$, and $j = 0, \ldots, k$.
Termination is expressed, using a parameter $\delta > 0$, by the commitment

$$C_0 \equiv time \in \langle T, T + \delta \rangle.$$

The module first, i.e. before a certain time Δ_0^i, applies its own arbitration number to $link_i$. Thus we have the commitment

$$C_1^i \equiv (apply, link_i[j], b_j^i) \textbf{ in } [0, \Delta_0^i\rangle.$$

Next MOD_i applies a value to its link at least once every Δ^i time units until time T is reached. This is expressed by the commitment

$$C_2^i \equiv t \in [\Delta_0^i + \Delta^i, T\rangle \rightarrow (apply, link_i[j]) \textbf{ in } [t - \Delta^i, t].$$

Furthermore, MOD_i either applies its own arbitration number or 0 to the link, i.e.,

$$C_3^i \equiv (apply, link_i[j], v) \textbf{ at } t \rightarrow (v = b_j^i \vee v = 0).$$

Observe that if $b_j^i = 0$ then $(apply, link_i[j], v)$ **at** t implies $v = b_j^i$. If the module applies 0 to a link line, whereas its corresponding own value is not 0, then it has been beaten on a more significant position by a recent value of the bus. That is,

$$C_4^i \equiv (apply, link_i[j], 0) \textbf{ at } t \wedge b_j^i \neq 0 \rightarrow \exists l < j : (bus[l] > b_l^i) \textbf{ in } [t - \Delta^i, t\rangle.$$

If the module applies its own non-zero value to a link line, then it is either in its initialization period (before time Δ_0^i) or it has not been beaten on a more significant position by a recent value of the bus. Thus we have the commitment

$$C_5^i \equiv (apply, link_i[j], b_j^i) \textbf{ at } t \wedge b_j^i \neq 0 \rightarrow$$
$$((t < \Delta_0^i) \vee (\forall l < j : (bus[l] \leq b_l^i) \textbf{ in } [t - \Delta^i, t\rangle)).$$

Note that all modules perform this protocol concurrently, possibly with different speed, without any synchronization, and in general starting at different times. In our compositional approach we first abstract from the implementation of modules by reasoning on the basis of their specifications only. We prove that these specifications lead to the required properties and determine the setting time T, when the number of the winner is on the bus, using the parameters for the first apply action (i.e. Δ_0^i) and the time between successive apply actions (i.e. Δ^i). With the assertions above, we can specify MOD_i by

$$\langle\!\langle time \in [0, \delta_0^i\rangle \wedge \neg(apply, link_i) \textbf{ at } [0, time\rangle\rangle\!\rangle \ MOD_i \ \langle\!\langle Spec_i\rangle\!\rangle$$

where $Spec_i \equiv C_0 \wedge C_{12345}^i$ with $C_{12345}^i \equiv \forall t < time \forall j \leq k : C_1^i \wedge C_2^i \wedge C_3^i \wedge C_4^i \wedge C_5^i$. Note that we need $\neg(apply, link_i)$ **at** $[0, time\rangle$ in the assumption of MOD_i, because otherwise C_4^i and C_5^i need not be valid at a point of time before the start of the program.

In table 1 we give an example of an execution of the protocol. Above the horizontal

Table 1. Example execution

MOD_0	MOD_1	MOD_2	MOD_3	MOD_4	bus
5724	4899	5677	5719	5723	0000
5724		5677	5719		0000
read	4899		read	5723	5779
		read		read	5899
5720	read		5710		5899
read	4000	5600	read	5700	5899
	read	read		read	5720
5700			5700		5720
read	4000	5600	read	5723	5700
		read		read	5723
5724	read		5719		5723
read	4000	5600	read	5723	5729
	read	read		read	5729
5724			5710		5729
read	4000	5600	read	5723	5724
	read	read		read	5724
5724			5710		5724
read	4000	5600		5723	5724

double line we give the arbitration numbers of five competing modules. Below this line we describe the actions of each module: a "read" when the module reads the value of the bus and a particular number when the module applies this number to its link. The last column gives the value of the bus, using the convention that an apply action affects the value of the bus on the next line. For instance, by the second read action MOD_0 reads 5899 and hence it applies 5700 the next time. Note that, to allow a concise representation, we have chosen a very simple execution, but in general actions of modules may overlap, leading to a large number of possible states.

To prove stability of the algorithm, we first present a lemma which expresses that

0. line 0 of $link_i$ always carries arbitration value b_0^i of MOD_i,
1. if a certain bus line is always greater than the corresponding arbitration number of a module after a certain point in time then, after a delay of $2\Delta^i$, all higher numbered links of the module will always carry value 0, and
2. if for a certain arbitration value no lower numbered bus line is greater after a certain point in time then, after a delay of $2\Delta^i$, the corresponding link line will always carry this arbitration value.

Lemma 3.1 (Properties Modules) For $i = 0, \ldots, n$, $Spec_i$ implies the following properties, for $j \in \{0, \ldots, k\}$ and $t \in TIME$,

(P0i) $(link_i[0] = b_0^i)$ **at** $[\Delta_0^i, time\rangle$

(P1i) $(bus[j] > b_j^i)$ **at** $[t, time\rangle \wedge t \in [\Delta_0^i, T - 2\Delta^i\rangle \rightarrow$
$\forall l > j : (link_i[l] = 0)$ **at** $[t + 2\Delta^i, time\rangle$

(P2i) $\forall l < j : (bus[l] \leq b_l^i)$ **at** $[t, time\rangle \wedge\ t \in [\Delta_0^i, T - 2\Delta^i\rangle \rightarrow$
$(link_i[j] = b_j^i)$ **at** $[t + 2\Delta^i, time\rangle$

Proof of (P0i): Note that $\neg(\exists l < 0 : (bus[l] > b_l^i)$ **in** $[t - \Delta^i, t\rangle)$, for all t, since l ranges over $I\!N$. Hence, by C_4^i, $\neg((apply, link_i[0], 0)$ **at** $t \wedge b_0^i \neq 0)$, for all t. Thus, using C_3^i, we have for all $t \in [0, time\rangle$ that $(apply, link_i[0], v)$ **at** t implies $v = b_0^i$. From C_1^i we obtain $(apply, link_i[0], b_0^i)$ **in** $[0, \Delta_0^i\rangle$ and hence, by lemma 2.2, $(link_i[0] = b_0^i)$ **at** $[\Delta_0^i, time\rangle$. ∎

Proof of (P1i): Note that (P1i) trivially holds if $t + 2\Delta^i \geq time$. Now suppose $t + 2\Delta^i < time$. Assume $(bus[j] > b_j^i)$ **at** $[t, time\rangle$ and $t \in [\Delta_0^i, T - 2\Delta^i\rangle$. Consider $l > j$. Let $t_1 \in [t + \Delta^i, time\rangle$. Since $t_1 - \Delta^i \geq t + \Delta^i - \Delta^i = t$ and $j < l$, the assumption $(bus[j] > b_j^i)$ **at** $[t, time\rangle$ leads to $\exists l_0 < l : (bus[l_0] > b_{l_0}^i)$ **at** $[t_1 - \Delta^i, t_1\rangle$. Thus, using lemma 2.1, $\neg(\forall l_0 < l : (bus[l_0] \leq b_{l_0}^i)$ **in** $[t_1 - \Delta^i, t_1\rangle)$. Together with $t \geq \Delta_0^i$, we obtain $\neg((t < \Delta_0^i) \vee (\forall l_0 < l : (bus[l_0] \leq b_{l_0}^i)$ **in** $[t_1 - \Delta^i, t_1\rangle))$. Since $t_1 < time$, assertion C_5^i leads to $\neg((apply, link_i[l], b_j^i)$ **at** $t_1 \wedge b_j^i \neq 0)$. By C_3^i, this implies, for all $t_1 \in [t + \Delta^i, time\rangle$, that if $(apply, link_i[l], v)$ **at** t_1 then $v = 0$. By assertion C_2^i, using $t + 2\Delta^i \in [\Delta_0^i, T\rangle$ and $t + 2\Delta^i < time$, we have

$(apply, link_i[l])$ **in** $[(t+2\Delta^i)-\Delta^i, t+2\Delta^i\rangle$. As shown above, the value 0 is applied, thus $(apply, link_i[l], 0)$ **in** $[t+\Delta^i, t+2\Delta^i\rangle$. Finally, by lemma 2.2 this leads to $(link_i[l]=0)$ **at** $[t+2\Delta^i, time\rangle$. ■

Proof of (P2i): Since (P2i)holds trivially holds if $t+2\Delta^i \geq time$, we can assume $t+2\Delta^i < time$. Suppose $\forall l < j : (bus[l] \leq b_l^i)$ **at** $[t, time\rangle$ and $t \in [\Delta_0^i, T-2\Delta^i\rangle$. Consider $t_1 \in [t+\Delta^i, time\rangle$. Since $\forall l < j : (bus[l] \leq b_l^i)$ **at** $[t_0 - \Delta^i, t_0\rangle$, we have, by lemma 2.1, $\neg(\exists l < j : (bus[l] > b_l^i)$ **in** $[t_1 - \Delta^i, t_1\rangle)$. Since $t_1 < time$, assertion C_4^i leads to $\neg((apply, link_i[j], 0)$ **at** $t_1 \wedge b_j^i \neq 0)$. Hence, using assertion C_3^i, we have, for all $t_1 \in [t+\Delta^i, time\rangle$, that $(apply, link_i[j], v)$ **at** t_1 implies $v = b_j^i$. By means of C_2^i, using $t+2\Delta^i \in [\Delta_0^i, T\rangle$ and $t+2\Delta^i < time$, we obtain the statement $(apply, link_i[j])$ **in** $[(t+2\Delta^i)-\Delta^i, t+2\Delta^i\rangle$ i.e. $(apply, link_i[j], b_j^i)$ **in** $[t+\Delta^i, t+2\Delta^i\rangle$, and hence, by lemma 2.2, $(link_i[j] = b_j^i)$ **at** $[t+2\Delta^i, time\rangle$. ■

The aim is to determine a constant T such that

$$\langle\langle time = 0\rangle\rangle \; MOD_0 \parallel \cdots \parallel MOD_n \; \langle\langle W_0 \wedge W_1\rangle\rangle$$

with $W_0 \equiv time \in \langle T, T+\delta\rangle$

$$W_1 \equiv \forall m \leq n : \\ (\forall i \neq m : val(b^m) > val(b^i)) \rightarrow (\forall j \leq k : (bus[j] = b_j^m) \textbf{ at } [T, time\rangle).$$

Thus W_1 expresses that if module MOD_m has the highest arbitration number then any bus line j carries the value b_j^m at any point in the interval $[T, time\rangle$.
Note that the specification of MOD_i only refers to the interface of MOD_i itself. Hence by n applications of the parallel composition rule we obtain

$$\langle\langle \textstyle\bigwedge_{i=0}^n time \in [0, \delta_0^i\rangle \wedge \neg(apply, link_i) \textbf{ at } [0, time\rangle\rangle\rangle \\ MOD_0 \parallel \cdots \parallel MOD_n \\ \langle\langle \exists t_0, \ldots, t_n : time = max(t_0, \ldots, t_n) \wedge \\ \textstyle\bigwedge_{i=0}^n Spec_i[t_i/time] \wedge \neg(apply, link_i) \textbf{ at } [t_i, time\rangle\rangle\rangle.$$

Observe that the commitment of this formula implies

$$\exists t_0, \ldots, t_n : time = max(t_0, \ldots, t_n) \wedge \\ \textstyle\bigwedge_{i=0}^n t_i \in \langle T, T+\delta\rangle \wedge C_{12345}^i[t_i/time] \wedge \neg(apply, link_i) \textbf{ at } [t_i, time\rangle.$$

Recall that $C_{12345}^i[t_i/time]$ is equivalent to $\forall t < t_i \, \forall j \leq k : C_1^i \wedge \ldots \wedge C_5^i$.
From $\neg(apply, link_i)$ **at** $[t_i, time\rangle$ we obtain $\forall t \in [t_i, time\rangle \, \forall j \leq k : C_3^i \wedge C_4^i \wedge C_5^i$, and thus $\forall t < time \, \forall j \leq k : C_3^i \wedge C_4^i \wedge C_5^i$. Further C_1^i does not depend on t, and $\forall t < t_i \, \forall j \leq k : C_2^i$ implies $\forall t < time \, \forall j \leq k : C_2^i$, since $time \geq t_i > T$. Hence the commitment leads to $time \in \langle T, T+\delta\rangle \wedge \bigwedge_{i=0}^n C_{12345}^i$. Since $time = 0$ implies $\bigwedge_{i=0}^n time \in [0, \delta_0^i\rangle \wedge \neg(apply, link_i)$ **at** $[0, time\rangle$, the consequence rule leads to

$$\langle\langle time = 0\rangle\rangle \; MOD_0 \parallel \cdots \parallel MOD_n \; \langle\langle C_0 \wedge \textstyle\bigwedge_{i=0}^n C_{12345}^i\rangle\rangle.$$

Since $W_0 \equiv C_0$, it remains to determine T and to prove W_1. Therefore, assume $\bigwedge_{i=0}^n C_{12345}^i$. To determine T, we use a list $T_0, \ldots, T_k$ of time points. The values of these time points will be derived during the proof of W_1, and finally T_k yields the value of T. Let m be such that $\forall i \neq m : val(b^m) > val(b^i)$. Define, for $\alpha = 0, \ldots, k$,

$$Stable(\alpha) \equiv \forall j \leq \alpha : (bus[j] = b^m_j) \textbf{ at } [T_\alpha, time\rangle.$$

We prove, by induction on α, that $Stable(\alpha)$ holds for all $\alpha \in \{0, \ldots, k\}$.

Proof: Let $MAX\Delta_0 = max(\Delta^0_0, \ldots, \Delta^n_0)$ and $MAX\Delta = max(\Delta^0, \ldots, \Delta^n)$.

Basic Step

For all $i \in \{0, \ldots, n\}$ we have, by (P0i), $(link_i[0] = b^i_0) \textbf{ at } [\Delta^i_0, time\rangle$.
Thus, for $t \in [\Delta^i_0, time\rangle$, we obtain by the bus property, $bus[0] \textbf{ at } (t + TB_0) = max(link_0[0] \textbf{ at } t, \ldots, link_n[0] \textbf{ at } t) = max(b^0_0, \ldots, b^n_0) = b^m_0$.
Hence $(bus[0] = b^m_0) \textbf{ at } [MAX\Delta_0 + TB_0, time\rangle$, and thus $Stable(0)$ with

$$T_0 = MAX\Delta_0 + TB_0. \tag{D1}$$

Induction Step

Suppose $\alpha < k$ and $Stable(\alpha)$. We prove $Stable(\alpha + 1)$. Consider $i \in \{0, \ldots, n\}$, $i \neq m$. By (A2) there exists an $l_i \leq k$ such that $b^m_{l_i} > b^i_{l_i}$ and $\forall j < l_i : b^m_j = b^i_j$.
Consider the cases $l_i \leq \alpha$ and $l_i \geq \alpha + 1$.

- Let $l_i \leq \alpha$. By $Stable(\alpha)$ we have $(bus[l_i] = b^m_{l_i}) \textbf{ at } [T_\alpha, time\rangle$.
  Since $b^i_{l_i} < b^m_{l_i}$ this leads to $(bus[l_i] > b^i_{l_i}) \textbf{ at } [T_\alpha, time\rangle$.
  By (P1i) we obtain $\forall j > l_i : (link_i[j] = 0) \textbf{ at } [T_\alpha + 2\Delta^i, time\rangle$, provided
  $$T_\alpha \in [\Delta^i_0, T - 2MAX\Delta\rangle. \tag{R1}$$
  Since $\alpha + 1 > l_i$, this leads to $(link_i[\alpha + 1] = 0) \textbf{ at } [T_\alpha + 2\Delta^i, time\rangle$.
  Thus by (A1), $(link_i[\alpha + 1] \leq b^m_{\alpha+1}) \textbf{ at } [T_\alpha + 2\Delta^i, time\rangle$, and hence
  $$(link_i[\alpha + 1] \leq b^m_{\alpha+1}) \textbf{ at } [T_\alpha + 2MAX\Delta, time\rangle, \text{ for } i \neq m. \tag{a}$$

- If $l_i \geq \alpha + 1$ then $b^i_{\alpha+1} \leq b^m_{\alpha+1}$. By C^i_3, we obtain
  $(link_i[\alpha + 1] \leq b^i_{\alpha+1}) \textbf{ at } [0, time\rangle$, and thus
  $$(link_i[\alpha + 1] \leq b^m_{\alpha+1}) \textbf{ at } [0, time\rangle, \text{ for } i \neq m. \tag{b}$$

By C^m_3 we have

$$(link_m[\alpha + 1] \leq b^m_{\alpha+1}) \textbf{ at } [0, time\rangle. \tag{c}$$

From (a), (b), and (c) we obtain, for all $i \in \{0, \ldots, n\}$,

$$(link_i[\alpha + 1] \leq b^m_{\alpha+1}) \textbf{ at } [T_\alpha + 2MAX\Delta, time\rangle. \tag{d}$$

By (P2m), using $Stable(\alpha)$, and provided

$$T_\alpha + 2MAX\Delta + TB_{\alpha+1} \in [\Delta^i_0, T - 2MAX\Delta\rangle \tag{R2}$$

we obtain
$(link_m[\alpha + 1] = b^m_{\alpha+1}) \textbf{ at } [T_\alpha + 2MAX\Delta + TB_{\alpha+1} + 2\Delta^m, time\rangle$, and thus
$(link_m[\alpha + 1] = b^m_{\alpha+1}) \textbf{ at } [T_\alpha + 4MAX\Delta + TB_{\alpha+1}, time\rangle$.
Together with (d) this leads, by the bus property, to
$(bus[\alpha + 1] = b^m_{\alpha+1}) \textbf{ at } [T_\alpha + 4MAX\Delta + 2TB_{\alpha+1}, time\rangle$.
Hence we have $Stable(\alpha + 1)$ with

$$T_{\alpha+1} = T_\alpha + 4MAX\Delta + 2TB_{\alpha+1}. \tag{D2}$$

■

Thus we have $Stable(\alpha)$, for all $\alpha \in \{0, \ldots, k\}$, provided the requirements (R1) and (R2) are fulfilled, and with $T_0, \ldots, T_k$ as defined by (D1) and (D2). From (D2) we obtain $T_\alpha = T_0 + 4\alpha MAX\Delta + \Sigma_{j=1}^{\alpha} 2TB_j$, and thus with (D1),

$$T_\alpha = MAX\Delta_0 + TB_0 + 4\alpha MAX\Delta + \Sigma_{j=1}^{\alpha} 2TB_j.$$

Hence we obtain W_1, that is, for all m,

$$(\forall i \neq m : val(b^m) > val(b^i)) \rightarrow (\forall j \leq k : (bus[j] = b_j^m) \textbf{ at } [T, time))$$

with $T = T_k$, that is,

$$T = MAX\Delta_0 + TB_0 + 4kMAX\Delta + \Sigma_{j=1}^{k} 2TB_j.$$

Finally, observe that with this definition of T the requirements (R1) and (R2) are fulfilled.

4. Programming Language

Although in the IEEE 896 Futurebus specification [8] the modules are implemented in hardware we derive here a software implementation of the modules from their specification. Therefore we consider a programming language in which concurrent processes can read the value of a common bus and they can apply a value to a link to the bus. Real-time is incorporated in the language by means of delay statements and the possibility to read local timers. In section 4.1 we give the syntax of this language and the informal meaning of statements. The basic timing assumptions, which are used to reason about real-time properties of programs, are described in section 4.2.

4.1. *Syntax and Informal Meaning*

Recall that the sets VAR, VAL, and $LINK$ have been introduced in section 2.1. Let $CLOCK$ be a nonempty set of names of local timers. The syntax of our programming language is given in table 2, with $N \in \mathbb{N}$, $N \geq 1$, $link \in LINK$, $clock \in CLOCK$, $x, x_0, \ldots, x_n \in VAR$, and $\vartheta \in VAL$.

Table 2. Syntax of the Programming Language

Expression	$e ::=$	$\vartheta \mid x \mid e_1 + e_2 \mid e_1 - e_2 \mid e_1 \times e_2$
Boolean Expression	$b ::=$	$e_1 = e_2 \mid e_1 < e_2 \mid \neg b \mid b_1 \vee b_2$
Statement	$S ::=$	$x := e \mid x := clock \mid \textbf{delay } e \mid$
		$read(x_0, \ldots, x_k) \mid apply(link, e_0, \ldots, e_k) \mid$
		$S_1 ; S_2 \mid G \mid *G \mid S_1 \| S_2$
Guarded Command	$G ::=$	$[\, [\!]_{i=1}^{N} b_i \rightarrow S_i]$

Informally, the statements of this language have the following meaning.
<u>Atomic statements</u>

- Assignment $x := e$ assigns the value of expression e to the variable x.

- $x := clock$ reads the value from local timer *clock* and assigns this value to x.
- **delay** e suspends execution for (the value of) e time units. If e yields a negative value then **delay** e terminates immediately.
- $read(x_0, \ldots, x_k)$ is used to read the value of the bus and assign the value of line j to x_j, for $j = 0, \ldots, k$.
- $apply(link, e_0, \ldots, e_k)$ is used to apply the value of e_j to line j of *link*, i.e., to $link[j]$, for $j = 0, \ldots, k$.

Compound statements

- $S_1; S_2$ indicates sequential composition: first execute S_1, and continue with the execution of S_2 if and when S_1 terminates.
- Guarded command $[\|_{i=1}^{N} b_i \rightarrow S_i]$. If none of the b_i evaluate to true then this guarded command terminates after evaluation of the booleans. Otherwise, nondeterministically select one of the b_i that evaluates to true and execute the corresponding statement S_i.
- Iteration $*G$ indicates repeated execution of guarded command G as long as at least one of the guards is open. When none of the guards is open $*G$ terminates.

Parallel composition and the sets $var(S)$ and $lnk(S)$ have been introduced already in section 2.1.

4.2. *Basic Timing Assumptions*

In our proof system the correctness of a program with respect to a specification, which may include timing constraints, is verified relative to assumptions about:

- The execution time of atomic statements. In this paper we use (nonnegative) parameters representing the execution time, i.e., we assume that
  - there exists a parameter T_a such that each assignment of the form $x := e$ takes T_a time units;
  - there exist parameters T_r and $T_w > 0$ such that each read action takes T_r time units, and each apply action takes T_w time units;
  - there exists a parameter T_c such that each assignment of the form $x := clock$ takes T_c time units.
- The overhead time required for compound programming constructs. Here we assume that there exists a parameter $T_g > 0$ such that the evaluation of guards in a guarded command takes T_g time units. There is no overhead for other compound statements.

5. Extending the Formal Framework

In this chapter the formalism of chapter 2 is extended to deal with the programming language of the previous chapter. In section 5.1 we extend the assertion language of section 2.2 to deal with local timers. Then we formulate a compositional proof system for our triples, axiomatizing the programming language. Recall that the consequence rule and the parallel composition rule have been given in section 2.4. First we give in section 5.2 rules and axioms that are generally applicable to any statement. Next in section 5.3 we axiomatize the programming language by giving rules and axioms for atomic statements and compound programming constructs.

5.1. Extension Assertion Language

To axiomatize local timers and their properties, we introduce the primitive

- $clock(exp)$ to denote the value of local timer $clock$ at time exp.

Local Clock Property

In this paper we assume that we can approximate durations by means of local timers. That is, the difference of two clock readings is a reasonable approximation of the elapsed real-time. Suppose local timers range over a discrete domain $CVAL$. Let u be the smallest unit of local clocks, that is, for any $T \in CVAL$, $\exists n \in I\!N : T = nu$. Further, for $T \in CVAL$, let $tick_{clock}(T)$ be the point in real-time where local timer $clock$ changes its value to T, i.e., $tick_{clock}(T) = inf\{t \in TIME \mid clock(t) = T\}$. The duration between ticks of a local timer is usually called the granularity g, that is, $g \approx tick_{clock}(T+u) - tick_{clock}(T)$. In general, this is not an equality since local clocks might have a certain drift r, and we have

$g(1-r) \leq tick_{clock}(T+u) - tick_{clock}(T) \leq g(1+r)$.

Then for $T_1, T_2 \in CVAL$, $T_1 \leq T_2$ implies

$$g(1-r)(T_2 - T_1) \leq (tick_{clock}(T_2) - tick_{clock}(T_1))u \leq g(1+r)(T_2 - T_1).$$

Note that $tick_{clock}(clock(t)) \leq t \leq tick_{clock}(clock(t) + u)$. Further assume that local timers are monotonic, i.e., $t_1 \leq t_2 \rightarrow clock(t_1) \leq clock(t_2)$. Then observe that for $t_1 \leq t_2$, $(t_2 - t_1)u \leq (tick_{clock}(clock(t_2) + u) - tick_{clock}(clock(t_1)))u \leq g(1+r)(clock(t_2)+u-clock(t_1))$. Similarly, $(t_2-t_1)u \geq g(1-r)(clock(t_2)-clock(t_1)-u)$. Thus we have the property

$$\boxed{t_1 \leq t_2 \quad \text{implies} \quad \frac{(t_2-t_1)u}{g(1+r)} - u \;\leq\; clock(t_2) - clock(t_1) \;\leq\; \frac{(t_2-t_1)u}{g(1-r)} + u}$$

Discussion For other applications one might need other assumptions about local timers. For instance, often local clocks are internally synchronized, that is, there exists a δ such that for any two timers $clock_1$ and $clock_2$,

$\forall t < \infty : |clock_1(t) - clock_2(t)| < \delta$.
Another possibility, which implies the previous one, is to assume that local clocks are externally synchronized and that they deviate less than ε from our global notion of time. Formally,
$\forall t < \infty : |clock(t) - t| < \varepsilon$.
End Discussion

5.2. *General Rules and Axioms*

We give three axioms to deduce invariance properties. The first axiom expresses that an assumption which satisfies certain restrictions remains valid during the execution of any program.

Axiom 5.1 (Initial Invariance) $\langle\langle p\rangle\rangle\ S\ \langle\langle p\rangle\rangle$

provided p *does not refer to the special variable* $time$ *or program variables (i.e., the variable* $time$ *does not occur in* p *and* $var(p) = \emptyset$*).*

In the next invariance axiom we express that the value of a program variable which does not occur in a program S is not affected by a terminating computation of S.

Axiom 5.2 (Variable Invariance) $\langle\langle p\rangle\rangle\ S\ \langle\langle time < \infty \rightarrow p\rangle\rangle$

provided the special variable $time$ *does not occur in* p *and* $p \notin var(S)$.

The link invariance axiom below expresses that during the execution of a program S no apply actions are performed to links that do not occur in S.

Axiom 5.3 (Link Invariance) $\langle\langle time = t_0\rangle\rangle\ S\ \langle\langle \neg(apply, link)\ \textbf{at}\ [t_0, time\rangle\rangle\rangle$

provided $link$ *does not occur in* S, *i.e.,* $link \notin lnk(S)$.

The next axiom expresses that when a program follows a nonterminating computation it does not affect this computation.

Axiom 5.4 (Nontermination) $\langle\langle p \wedge time = \infty\rangle\rangle\ S\ \langle\langle p \wedge time = \infty\rangle\rangle$

The substitution rule allows us to replace a logical variable in the assumption by any arbitrary expression if this variable does not occur in the commitment.

Rule 5.1 (Substitution)
$$\frac{\langle\langle p\rangle\rangle\ S\ \langle\langle q\rangle\rangle}{\langle\langle p[exp/t]\rangle\rangle\ S\ \langle\langle q\rangle\rangle}$$

provided t *does not occur free in* q.

The proof system contains the usual conjunction rule.

Rule 5.2 (Conjunction)
$$\frac{\langle\langle p_1\rangle\rangle\ S\ \langle\langle q_1\rangle\rangle, \langle\langle p_2\rangle\rangle\ S\ \langle\langle q_2\rangle\rangle}{\langle\langle p_1 \wedge p_2\rangle\rangle\ S\ \langle\langle q_1 \wedge q_2\rangle\rangle}$$

5.3. Axiomatization of Programming Constructs

Next we give rules and axioms for the atomic statements and the compound constructs of the programming language. We assume that all logical variables which are introduced in the rules are fresh.
In the rule for an assignment $x := e$ we express that to obtain commitment q the assumption $q[e/x, time + T_a/time] \wedge time < \infty$ is required (i.e., this is the weakest assumption). Note that, in addition to the classical rule, we also update the variable $time$ to express that the termination time equals the initial time plus T_a time units.

Axiom 5.5 (Assignment) $\langle\langle q[e/x, time + T_a/time] \wedge time < \infty \rangle\rangle \; x := e \; \langle\langle q \rangle\rangle$

Example: We derive
$\langle\langle x = 5 \wedge time = 6 \wedge (apply, link_4[2], 0) \textbf{ at } 3 \rangle\rangle$
$x := x + 7$
$\langle\langle x = 12 \wedge time = 6 + T_a \wedge (apply, link_4[2], 0) \textbf{ at } 3 \rangle\rangle$.
The assignment axiom leads to
$\langle\langle x + 7 = 12 \wedge time + T_a = 6 + T_a \wedge (apply, link_4[2], 0) \textbf{ at } 3 \wedge time < \infty \rangle\rangle$
$x := x + 7$
$\langle\langle x = 12 \wedge time = 6 + T_a \wedge (apply, link_4[2], 0) \textbf{ at } 3 \rangle\rangle$.
Then the consequence rule (see section 2.4) yields the required triple, since
$x = 5 \wedge time = 6 \wedge (apply, link_4[2], 0) \textbf{ at } 3 \rightarrow$
$x + 7 = 12 \wedge time + T_a = 6 + T_a \wedge (apply, link_4[2], 0) \textbf{ at } 3 \wedge time < \infty$. □

Further we axiomatize local clock readings and delay actions.

Axiom 5.6 (Clock)

$$\langle\langle q[clock(time)/x, time + T_c/time] \wedge time < \infty \rangle\rangle \; x := clock \; \langle\langle q \rangle\rangle$$

Axiom 5.7 (Delay) $\langle\langle q[time + max(0, e)/time] \wedge time < \infty \rangle\rangle \; \textbf{delay } e \; \langle\langle q \rangle\rangle$

In the rule for a read action $read(x_0, \ldots, x_k)$ we first replace $time$ in assumption $p \wedge time < \infty$ by t_0. Then we express that this read action assigns the value of $bus[j]$ at the starting time (represented by t_0) to variable x_j, for $j \leq k$, and then terminates after T_r time units.

Rule 5.3 (Read)

$$\frac{\begin{array}{r}(p \wedge time < \infty)[t_0/time] \wedge \forall j \leq k : x_j = (bus[j] \textbf{ at } t_0) \wedge \\ time = t_0 + T_r \rightarrow q\end{array}}{\langle\langle p \wedge time < \infty \rangle\rangle \; read(x_0, \ldots, x_k) \; \langle\langle q \rangle\rangle}$$

provided $x_0, \ldots, x_k$ do not occur free in p.

The rule for an apply action $apply(link, e_0, \ldots, e_k)$ expresses that at the starting time (again represented by t_0) the value of e_j is applied to line $link[j]$. To be complete, we also assert that no apply action to $link$ starts between t_0 and the termination time. Note that for other atomic statements this can be derived by the link invariance axiom.

Rule 5.4 (Apply)

$$\frac{\begin{array}{l}(p \wedge time < \infty)[t_0/time] \wedge \forall j \leq k : (apply, link[j], e_j) \textbf{ at } t_0 \wedge \\ \qquad\qquad \neg(apply, link) \textbf{ at } \langle t_0, time\rangle \wedge time = t_0 + T_w \rightarrow q\end{array}}{\langle\!\langle p \wedge time < \infty\rangle\!\rangle \ apply(link, e_0, \ldots, e_k) \ \langle\!\langle q\rangle\!\rangle}$$

The inference rule for sequential composition is similar to the classical rule for Hoare triples.

Rule 5.5 (Sequential Composition) $\dfrac{\langle\!\langle p\rangle\!\rangle \ S_1 \ \langle\!\langle r\rangle\!\rangle, \quad \langle\!\langle r\rangle\!\rangle \ S_2 \ \langle\!\langle q\rangle\!\rangle}{\langle\!\langle p\rangle\!\rangle \ S_1; S_2 \ \langle\!\langle q\rangle\!\rangle}$

Furthermore the proof system contains rules for guarded commands and iteration. For a guarded command $G \equiv [\,[\!]_{i=1}^{N} b_i \rightarrow S_i]$, first define $b_G = \bigvee_{i=1}^{N} b_i$.

Rule 5.6 (Guarded Command)

$$\frac{\begin{array}{l}\langle\!\langle p \wedge \neg b_G\rangle\!\rangle \textbf{ delay } T_g \ \langle\!\langle q\rangle\!\rangle \\ \langle\!\langle p \wedge b_i\rangle\!\rangle \textbf{ delay } T_g \ ; \ S_i \ \langle\!\langle q_i\rangle\!\rangle, for\ all\ i \in \{1, \ldots, n\}\end{array}}{\langle\!\langle p\rangle\!\rangle \ [\,[\!]_{i=1}^{n} b_i \rightarrow S_i] \ \langle\!\langle q \vee \bigvee_{i=1}^{n} q_i\rangle\!\rangle}$$

In the rule for the iteration construct we have the two traditional clauses of Hoare logic, which establish an invariant I and lead to commitment q upon termination. Further there are a few additional conditions to deal with nonterminating computations.

Rule 5.7 (Iteration)

$$\frac{\begin{array}{l}\langle\!\langle I \wedge b_G\rangle\!\rangle \ G \ \langle\!\langle I\rangle\!\rangle \\ \langle\!\langle I \wedge \neg b_G\rangle\!\rangle \ G \ \langle\!\langle q\rangle\!\rangle \\ I \rightarrow I_0, var(I_0) = \emptyset \\ (\forall t_1 < \infty \, \exists t_2 > t_1 : I_0[t_2/time]) \rightarrow q_{nt}\end{array}}{\langle\!\langle I\rangle\!\rangle \ * G \ \langle\!\langle (q_{nt} \wedge time = \infty) \vee q\rangle\!\rangle}$$

To give an informal indication why assertion q_{nt} holds for an infinite computation (satisfying $time = \infty$), consider a nonterminating computation of $*G$, starting in a state satisfying I. Then there are two possibilities:

- It is obtained from an infinite sequence of terminating computations of G. For all these computations b_G is true initially. Hence, using the first clause of the rule, we can prove by induction that all these computations satisfy I. By $I \rightarrow I_0$, also I_0 holds on all these computations. Since each computation of G takes at least T_g time units, after any point in time there exists a point at which I_0 holds. Thus $\forall t_1 < \infty \, \exists t_2 > t_1 : I_0[t_2/time]$, and hence q_{nt} by the fourth condition.

- It is obtained from a finite number of terminating computations followed by a nonterminating computation of G. Then, as in the previous point, we can prove that $I \wedge b_G$ holds in the initial state of this last computation. Thus, using the first condition and the fact that it is a nonterminating computation, $I \wedge time = \infty$ holds for this computation. Hence $I_0 \wedge time = \infty$, which implies $\forall t_1 < \infty \, \exists t_2 > t_1 : I_0[t_2/time]$, and then the fourth condition leads to q_{nt}.

Observe that the implication $I \rightarrow I_0$ and the condition $var(I_0) = \emptyset$ are required to avoid assertions about the state, i.e. the values of the variables, of a nonterminating computation.

The rules above axiomatize the programming language, but they are not always easy to use. Often it is more convenient to derive new rules for certain applications. As an example we derive here a rule for clock assignments which will be used in the verification of the protocol.

Derived Rule 5.8 (Clock)

$$\frac{\begin{array}{l}(p \wedge time < \infty)[t_0/time, v/x] \wedge x = clock(t_0) \wedge \\ \qquad \neg(apply, link) \ \mathbf{at} \ [t_0, time\rangle \wedge time = t_0 + T_c \rightarrow q\end{array}}{\langle\langle p \wedge time < \infty \rangle\rangle \ x := clock \ \langle\langle q \rangle\rangle}$$

for any $link \in LINK$.

Proof: We show that this rule can be derived from the basic rules and axioms. Let $q_0 \equiv (p \wedge time < \infty)[t_0/time, v/x] \wedge x = clock(t_0) \wedge \, time = t_0 + T_c$.
By the clock axiom we can derive
$\langle\langle q_0[clock(time)/x, time + T_c/time] \wedge time < \infty \rangle\rangle \ x := clock \ \langle\langle q_0 \rangle\rangle$.
Note that $q_0[clock(time)/x, time + T_c/time]$ is equivalent to
$(p \wedge time < \infty)[t_0/time, v/x] \wedge clock(time) = clock(t_0) \wedge \, time + T_c = t_0 + T_c$.
Thus $(p \wedge time < \infty)[t_0/time, v/x] \wedge time = t_0$ implies
$q_0[clock(time)/x, time + T_c/time] \wedge time < \infty$, and hence by the consequence rule
$\langle\langle (p \wedge time < \infty)[t_0/time, v/x] \wedge time = t_0 \rangle\rangle \ x := clock \ \langle\langle q_0 \rangle\rangle$.
By the link invariance rule we can derive
$\langle\langle time = t_0 \rangle\rangle \ x := clock \ \langle\langle \neg(apply, link) \ \mathbf{at} \ [t_0, time\rangle \rangle\rangle$.
Hence the conjunction rule leads to
$\langle\langle (p \wedge time < \infty)[t_0/time, v/x] \wedge time = t_0 \rangle\rangle \ x := clock$
$\langle\langle q_0 \wedge \neg(apply, link) \ \mathbf{at} \ [t_0, time\rangle \rangle\rangle$.
By the hypothesis of rule 5.8 we have $q_0 \wedge \neg(apply, link) \ \mathbf{at} \ [t_0, time\rangle \rightarrow q$, thus
$\langle\langle (p \wedge time < \infty)[t_0/time, v/x] \wedge time = t_0 \rangle\rangle \ x := clock \ \langle\langle q \rangle\rangle$.
Using the general assumption that logical variables are fresh, t_0 and v do not occur in q. Hence by the substitution rule we can replace t_0 by $time$ and v by x in the assumption, leading to
$\langle\langle p \wedge time < \infty \rangle\rangle \ x := clock \ \langle\langle q \rangle\rangle$. ■

6. Implementation of the Modules

In this chapter we derive a software implementation of a module according to the specification of chapter 3. Since we deal with one module now, all indices i are omitted. Thus the aim is to find a program MOD that satisfies the specification

$$\langle\langle p_0 \rangle\rangle \; MOD \; \langle\langle C_0 \wedge C_{12345} \rangle\rangle$$

with
$p_0 \equiv time \in [0, \delta_0\rangle \wedge \neg(apply, link)$ **at** $[0, time\rangle$,
$C_0 \equiv time \in \langle T, T + \delta\rangle$, and
$C_{12345} \equiv \forall t < time \; \forall j \leq k : C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge C_5$ where
$C_1 \equiv (apply, link[j], b_j)$ **in** $[0, \Delta_0\rangle$
$C_2 \equiv t \in [\Delta_0 + \Delta, T\rangle \rightarrow (apply, link[j])$ **in** $[t - \Delta, t]$
$C_3 \equiv (apply, link[j], v)$ **at** $t \rightarrow (v = b_j \vee v = 0)$
$C_4 \equiv (apply, link[j], 0)$ **at** $t \wedge b_j \neq 0 \rightarrow \exists l < j : (bus[l] > b_l)$ **in** $[t - \Delta, t\rangle$
$C_5 \equiv (apply, link[j], b_j)$ **at** $t \wedge b_j \neq 0 \rightarrow$
$((t < \Delta_0) \vee (\forall l < j : (bus[l] \leq b_l)$ **in** $[t - \Delta, t\rangle))$.

Henceforth, we use $C_{2345} \equiv \forall t < time \; \forall j \leq k : C_2 \wedge C_3 \wedge C_4 \wedge C_5$ and, similarly, $C_{345} \equiv \forall t < time \; \forall j \leq k : C_3 \wedge C_4 \wedge C_5$.

Implementation of MOD

As a first design step to implement MOD, we start reading a local timer. The value of this timer will be used to terminate the program in the required time interval as specified by C_0. Define
$I_{clock} \equiv \exists t \in [0, \delta_0\rangle : x_0 = clock(t)$, and
$r \equiv time \in [T_c, \delta_0 + T_c\rangle \wedge \neg(apply, link)$ **at** $[0, time\rangle \wedge I_{clock}$.
Then we can derive

$$\langle\langle p_0 \rangle\rangle \quad x_0 := clock \quad \langle\langle r \rangle\rangle$$

by the derived clock rule 5.8 and the consequence rule, since $p_0 \rightarrow p_0 \wedge time < \infty$ and $(p_0 \wedge time < \infty)[t_0/time, v/x_0] \wedge x_0 = clock(t_0) \wedge \neg(apply, link)$ **at** $[t_0, time\rangle \wedge$ $time = t_0 + T_c$ implies $t_0 \in [0, \delta_0\rangle \wedge \neg(apply, link)$ **at** $[0, t_0\rangle \wedge x_0 = clock(t_0) \wedge$ $\neg(apply, link)$ **at** $[t_0, time\rangle \wedge time = t_0 + T_c$ which implies r. Let

$$MOD \equiv x_0 := clock \; ; \; MOD_0 \qquad \text{(M)}$$

By the sequential composition rule this design step is justified if MOD_0 satisfies

$$\langle\langle r \rangle\rangle \; MOD_0 \; \langle\langle C_0 \wedge C_{12345} \rangle\rangle.$$

This is a typical example of a sequential top-down design step: we implement part of the program and show that the step is correct using a specification of the remainder.

Implementation of MOD_0

Given assertion C_1 in the commitment, MOD_0 should continue with the apply action $apply(link, b_0, \ldots, b_k)$. Let
$r_0 \equiv C_{2345} \wedge time < \delta_0 + T_c + T_w \wedge (apply, link)$ **at** $(time - T_w) \wedge I_{clock}$.
Then by the apply rule we can derive

$$\langle\!\langle r \rangle\!\rangle \;\; apply(link, b_0, \ldots, b_k) \;\; \langle\!\langle C_1 \wedge r_0 \rangle\!\rangle$$

provided

$$\Delta \geq T_w \tag{T0}$$

$$\Delta_0 \geq \delta_0 + T_c \tag{T1}$$

where (T0) is required to prove C_2 and (T1) is needed for C_1 and C_5.
Note that in the commitment r_0 we record information about the termination time and the time of the last apply action. Let

$$MOD_0 \equiv apply(link, b_0, \ldots, b_k)\,;\; MOD_1 \tag{M0}$$

By the sequential composition rule, this step is correct if MOD_1 satisfies

$$\langle\!\langle C_1 \wedge r_0 \rangle\!\rangle \; MOD_1 \; \langle\!\langle C_0 \wedge C_{12345} \rangle\!\rangle.$$

Using the definition of C_1 and the initial invariance axiom we have, for any program MOD_1,

$$\langle\!\langle C_1 \rangle\!\rangle \; MOD_1 \; \langle\!\langle C_1 \rangle\!\rangle.$$

Hence, by the conjunction and the consequence rule, it is sufficient to design MOD_1 such that it satisfies the specification

$$\langle\!\langle r_0 \rangle\!\rangle \; MOD_1 \; \langle\!\langle C_0 \wedge C_{2345} \rangle\!\rangle.$$

Implementation of MOD_1

The specification of MOD_1 expresses that we have to establish the termination condition C_0 while maintaining C_{2345} invariant. Hence we implement MOD_1 as an iteration that performs read and apply actions according to C_{2345} until, approximately, T time units have elapsed. Therefore we read the local timer *clock* regularly and compare its value with the initial value x_0. Suppose x contains the last value we have read from *clock*. Then by the local clock property we obtain some relation between $x - x_0$ and the real-time duration. This leads to an iteration with guard $x - x_0 \leq D$, using a parameter D which has to be determined later. Clearly we first have to initialize x by reading *clock* before the iteration is performed. Thus the idea is to implement MOD_1 as follows.

$$MOD_1 \equiv x := clock\;;\; *[x - x_0 \leq D \rightarrow MOD_2\;;\; x := clock] \tag{M1}$$

Let
$r_1 \equiv C_{2345} \wedge time < \delta_0 + 2T_c + T_w \wedge (apply, link)$ **at** $(time - T_w - T_c) \wedge$
$x = clock(time - T_c) \wedge I_{clock}$.
Then we can derive

$\langle\!\langle r_0 \rangle\!\rangle \; x := clock \; \langle\!\langle r_1 \rangle\!\rangle$

provided, to prove C_2,

$\Delta \geq T_w + T_c$ (T2)

Observe that C_{345} and I_{clock} are invariant by the following lemma (note that C_0 in r_1 implies $time < \infty$), which can be derived by the invariance axioms, the conjunction rule, and the consequence rule.

Lemma 6.1 If S does not contain *link* and $x_0 \notin var(S)$, then
$\langle\!\langle C_{345} \wedge I_{clock} \rangle\!\rangle \; S \; \langle\!\langle C_{345} \wedge (time < \infty \rightarrow I_{clock}) \rangle\!\rangle$.

Specification of MOD_2

Given commitment r_1 of the clock reading, MOD_2 should be such that

$$\langle\!\langle r_1 \rangle\!\rangle \; *[x - x_0 \leq D \rightarrow MOD_2 \; ; \; x := clock] \; \langle\!\langle C_0 \wedge C_{2345} \rangle\!\rangle.$$

To derive this formula we use the iteration rule with the assertions

$I \equiv C_{2345} \wedge time < T + \delta - T_g \wedge (apply, link) \textbf{ at } (time - T_w - T_c) \wedge$
$\quad x = clock(time - T_c) \wedge I_{clock}$,

$I_0 \equiv time < T + \delta - T_g$,

$q \equiv C_0 \wedge C_{2345}$, and

$q_{nt} \equiv false$.

Observe that if the requirements of the rule are fulfilled for these assertions, then we obtain

$\langle\!\langle I \rangle\!\rangle \; * \; [x - x_0 \leq D \rightarrow MOD_2 \; ; \; x := clock] \; \langle\!\langle q \rangle\!\rangle$

and thus the desired formula, since $r_1 \rightarrow I$, provided

$\delta_0 + 2T_c + T_w + T_g \leq T + \delta$ (T3)

Note that $I \rightarrow I_0$ and $var(I_0) = \emptyset$. By the iteration rule, it remains to prove

$$\langle\!\langle I \wedge x - x_0 \leq D \rangle\!\rangle \; [x - x_0 \leq D \rightarrow MOD_2 \; ; \; x := clock] \; \langle\!\langle I \rangle\!\rangle \tag{1}$$

$$\langle\!\langle I \wedge x - x_0 > D \rangle\!\rangle \; [x - x_0 \leq D \rightarrow MOD_2 \; ; \; x := clock] \; \langle\!\langle q \rangle\!\rangle \tag{2}$$

$$(\forall t_1 < \infty \; \exists t_2 > t_1 : I_0[t_2/time]) \rightarrow q_{nt} \tag{3}$$

Observe that that (3) holds, since

$(\forall t_1 < \infty \; \exists t_2 > t_1 : I_0[t_2/time]) \rightarrow$
$(\forall t_1 < \infty \; \exists t_2 > t_1 : (time < T + \delta - T_g)[t_2/time]) \rightarrow$
$(\forall t_1 < \infty \; \exists t_2 > t_1 : t_2 < T + \delta - T_g) \rightarrow false$, i.e., q_{nt}.

To prove (2), by the guarded command rule, it is sufficient to derive

$\langle\!\langle I \wedge x - x_0 > D \rangle\!\rangle$ **delay** $T_g \; \langle\!\langle q \rangle\!\rangle$.

We show that this formula can be derived, provided

$\Delta \geq T_w + T_c + T_g$ (T4)

$D \geq \frac{(T - T_g - T_c)u}{g(1-r)} + u$ (T5)

Proof: Note that $I \wedge x - x_0 > D$ implies $I \wedge x - x_0 > D \wedge time < \infty$.
By the delay and the link invariance axioms we have to show, similar to rule 5.8, $I[t_0/time] \wedge x - x_0 > D \wedge \neg(apply, link)$ **at** $[t_0, time\rangle \wedge time = t_0 + T_g \rightarrow q$.
First observe that C_{2345} in q follows from $C_{2345}[t_0/time] \wedge$ $(apply, link)$ **at** $(t_0 - T_w - T_c) \wedge \neg(apply, link)$ **at** $[t_0, time\rangle \wedge time = t_0 + T_g$, using (T4). It remains to show C_0, that is, $time > T$ and $time < T + \delta$.

- To prove $time > T$ observe that from $I[t_0/time]$ and $time = t_0 + T_g$ we obtain $x = clock(time - T_g - T_c)$ and there exists a $t \in [0, \delta_0\rangle$ such that $x_0 = clock(t)$. By the property of local timers we have
  $clock(time - T_g - T_c) - clock(t) \leq \frac{(time - T_g - T_c - t)u}{g(1-r)} + u$.
  Since $D < x - x_0 = clock(time - T_g - T_c) - clock(t)$ we obtain
  $\frac{(time - T_g - T_c - t)u}{g(1-r)} + u > D$.
  Thus $time > \frac{(D-u)g(1-r)}{u} + T_g + T_c + t$.
  By $t \geq 0$ this leads to $time > \frac{(D-u)g(1-r)}{u} + T_g + T_c$.
  Hence we have $time > T$ if $\frac{(D-u)g(1-r)}{u} + T_g + T_c \geq T$ which follows from (T5).
- To prove $time < T + \delta$ we use $t_0 < T + \delta - T_g$ from $I[t_0/time]$. Together with $time = t_0 + T_g$ this leads to $time < T + \delta - T_g + T_g = T + \delta$.

■

Hence it remains to find MOD_2 such that $[x - x_0 \leq D \rightarrow MOD_2 \; ; \; x := clock]$ satisfies (1). By the rule for guarded commands this implies that we should derive

$$\langle\!\langle I \wedge x - x_0 \leq D \rangle\!\rangle \; \textbf{delay} \; T_g \; ; \; MOD_2 \; ; \; x := clock \; \langle\!\langle I \rangle\!\rangle. \tag{4}$$

Observe that I implies $x = clock(time - T_c)$ and there exists a $t \in [0, \delta_0\rangle$ such that $x_0 = clock(t)$. Together with $x - x_0 \leq D$ and the properties of local timers we obtain $D \geq x - x_0 = clock(time - T_c) - clock(t) \geq \frac{(time - T_c - t)u}{g(1+r)} - u$.
Thus $time \leq \frac{(D+u)g(1+r)}{u} + T_c + t \leq \frac{(D+u)g(1+r)}{u} + T_c + \delta_0$. Define

$$T_s \equiv \frac{(D+u)g(1+r)}{u} + T_c + \delta_0. \tag{Def1}$$

Then, as shown above, $I \wedge x - x_0 \leq D$ implies $time \leq T_s$. Hence $I \wedge x - x_0 \leq D$ leads to $C_{2345} \wedge time \leq T_s \wedge (apply, link)$ **at** $(time - T_w - T_c)$. Let
$p_1 \equiv C_{2345} \wedge time \leq T_s + T_g \wedge (apply, link)$ **at** $(time - T_w - T_c - T_g)$.
Then, using (T4), we can derive

$$\langle\!\langle I \wedge x - x_0 \leq D \rangle\!\rangle \; \textbf{delay} \; T_g \; \langle\!\langle p_1 \rangle\!\rangle. \tag{5}$$

Let $q_1 \equiv C_{2345} \wedge time < T + \delta - T_g - T_c \wedge (apply, link)$ **at** $(time - T_w) \wedge I_{clock}$.
Then we can derive, using (T2),

$$\langle\!\langle q_1 \rangle\!\rangle \; x := clock \; \langle\!\langle I \rangle\!\rangle. \tag{6}$$

By the sequential composition rule, formula (4) can be derived from (5) and (6) if

$$\langle\!\langle p_1 \rangle\!\rangle \; MOD_2 \; \langle\!\langle q_1 \rangle\!\rangle.$$

Implementation of MOD_2

Clearly commitment q_1 of MOD_2 requires that the last action of MOD_2 should be an apply action. Thus, introducing a list of variables $w_0, \ldots, w_k$, let

$$MOD_2 \equiv MOD_3\ ;\ apply(link, w_0, \ldots, w_k) \tag{M2}$$

To obtain commitment q_1, the values of $w_0, \ldots, w_k$ should be such that C_3, C_4 and C_5 are satisfied. Hence the assumption of the apply action, which is the commitment of MOD_3, should express this. Therefore we define

$$\begin{aligned} BW(j) \equiv\ & (w_j = 0 \wedge b_j \neq 0 \rightarrow \exists l < j : (bus[l] > b_l) \textbf{ in } [time - \Delta, time\rangle)\wedge \\ & (w_j = b_j \wedge b_j \neq 0 \rightarrow \forall l < j : (bus[l] \leq b_l) \textbf{ in } [time - \Delta, time\rangle)\wedge \\ & (w_j = 0 \vee w_j = b_j) \end{aligned}$$

Let $q_2 \equiv C_{2345} \wedge time < T + \delta - T_g - T_c - T_w \wedge\ \forall j \leq k : BW(j) \wedge I_{clock}$.

Then for this apply action we can prove

$$\langle\langle q_2 \rangle\rangle\ apply(link, w_0, \ldots, w_k)\ \langle\langle q_1 \rangle\rangle.$$

It remains to implement MOD_3 such that

$$\langle\langle p_1 \rangle\rangle\ MOD_3\ \langle\langle q_2 \rangle\rangle.$$

Implementation of MOD_3

By lemma 6.1 we obtain

$$\langle\langle C_{345} \wedge I_{clock} \rangle\rangle\ MOD_3\ \langle\langle C_{345} \wedge (time < \infty \rightarrow I_{clock}) \rangle\rangle,$$

provided

$$MOD_3 \text{ does not contain } link \text{ and } x_0 \notin var(MOD_3). \tag{RM1}$$

Let $\hat{C}_2 \equiv \forall t < time : C_2$.

Using (RM1), by the conjunction and the consequence rule, it is sufficient to derive

$$\begin{aligned} & \langle\langle \hat{C}_2 \wedge time \leq T_s + T_g \wedge (apply, link) \textbf{ at } (time - T_w - T_c - T_g) \rangle\rangle \\ & \quad MOD_3 \\ & \langle\langle \hat{C}_2 \wedge time < T + \delta - T_g - T_c - T_w \wedge \forall j \leq k : BW(j) \rangle\rangle. \end{aligned}$$

Since $\forall j \leq k : BW(j)$ occurs in the commitment of MOD_3, it is clear by the definition of $BW(j)$ that MOD_3 should compute the values $w_0, \ldots, w_k$. To establish $BW(j)$ we have to know the status of the bus. Therefore the first action is to read the bus. Let

$$MOD_3 \equiv read(r_0, \ldots, r_k)\ ;\ MOD_4 \tag{M3}$$

For this read action we derive

$$\begin{aligned} & \langle\langle \hat{C}_2 \wedge time \leq T_s + T_g \wedge (apply, link) \textbf{ at } (time - T_w - T_c - T_g) \rangle\rangle \\ & \quad read(r_0, \ldots, r_k) \\ & \langle\langle \hat{C}_2 \wedge time \leq T_s + T_g + T_r \wedge (apply, link) \textbf{ at } (time - T_w - T_c - T_g - T_r)\wedge \\ & \ \ \forall j \leq k : (bus[j] \textbf{ at } time - T_r) = r_j \rangle\rangle. \end{aligned}$$

Define $X_1 \equiv T_s + T_g + T_r$ and $X_2 \equiv T_w + T_c + T_g + T_r$. Then MOD_4 must satisfy

$$\begin{aligned} & \langle\langle \hat{C}_2 \wedge time \leq X_1 \wedge (apply, link) \textbf{ at } (time - X_2)\wedge \\ & \ \ \forall j \leq k : (bus[j] \textbf{ at } time - T_r) = r_j \rangle\rangle \\ & \quad MOD_4 \\ & \langle\langle \hat{C}_2 \wedge time < T + \delta - T_g - T_c - T_w \wedge\ \forall j \leq k : BW(j) \rangle\rangle. \end{aligned}$$

Reducing the specification of MOD_4

Before we implement MOD_4 we first reduce the specification. Define
$RW(j) \equiv (w_j = 0 \wedge b_j \neq 0 \rightarrow \exists l < j : r_l > b_l) \wedge$
$(w_j = b_j \wedge b_j \neq 0 \rightarrow \forall l < j : r_l \leq b_l) \wedge$
$(w_j = 0 \vee w_j = b_j)$.
We show that it is sufficient to derive, for a certain time parameter T_{max},

$$\langle\langle time = t_0 \rangle\rangle \ MOD_4 \ \langle\langle time \leq t_0 + T_{max} \wedge \forall j \leq k : RW(j) \rangle\rangle. \tag{7}$$

By the initial invariance axiom and the consequence rule we can derive

$$\begin{array}{l} \langle\langle (\forall t < t_0 : C_2) \wedge t_0 \leq X_1 \wedge (apply, link) \ \mathbf{at} \ (t_0 - X_2) \wedge \\ \quad \forall j \leq k : (bus[j] \ \mathbf{at} \ t_0 - T_r) = r_j \rangle\rangle \\ \qquad MOD_4 \\ \langle\langle (\forall t < t_0 : C_2) \wedge t_0 \leq X_1 \wedge (apply, link) \ \mathbf{at} \ (t_0 - X_2) \wedge \\ \quad \forall j \leq k : (bus[j] \ \mathbf{at} \ t_0 - T_r) = r_j \rangle\rangle \end{array} \tag{8}$$

provided
MOD_4 does not change the variables $r_0, \ldots, r_k$. (RM2)
Further note that it is easy to prove

$$\langle\langle time = t_0 \rangle\rangle \ MOD_4 \ \langle\langle t_0 \leq time \rangle\rangle. \tag{9}$$

Then the conjunction rule applied to (7), (8), and (9) leads to the commitment
$t_0 \leq time \leq t_0 + T_{max} \wedge \forall j \leq k : RW(j) \wedge (\forall t < t_0 : C_2) \wedge t_0 \leq X_1 \wedge$
$(apply, link) \ \mathbf{at} \ (t_0 - X_2) \wedge \forall j \leq k : (bus[j] \ \mathbf{at} \ t_0 - T_r) = r_j$.
We can show that this implies the required commitment of MOD_4, i.e.,
$\hat{C}_2 \wedge time < T + \delta - T_g - T_c - T_w \wedge \forall j \leq k : BW(j)$, provided
$T_s + T_{max} + T_c + 2T_g + T_r + T_w < T + \delta$ (T6)
$T_{max} + T_w + T_c + T_g + T_r \leq \Delta$ (T7)

Then the consequence rule leads to

$$\begin{array}{l} \langle\langle time = t_0 \wedge (\forall t < t_0 : C_2) \wedge t_0 \leq X_1 \wedge (apply, link) \ \mathbf{at} \ (t_0 - X_2) \wedge \\ \quad \forall j \leq k : (bus[j] \ \mathbf{at} \ t_0 - T_r) = r_j \rangle\rangle \\ \qquad MOD_4 \\ \langle\langle \hat{C}_2 \wedge time < T + \delta - T_g - T_c - T_w \wedge \forall j \leq k : BW(j) \rangle\rangle. \end{array}$$

By the substitution rule, replacing t_0 in the assumption by *time*, and the consequence rule we obtain the required triple.

Implementation of MOD_4

Hence we implement MOD_4 according to (7). The formula $\forall j \leq k : RW(j)$ in the commitment of this formula suggests an iteration for which $\forall j < i : RW(j)$ is an invariant, using an auxiliary variable i, and which terminates with $i = k + 1$. The intuition is that in the $(i+1)^{\text{th}}$ execution of the body of the iteration we compare

r_i and b_i and assign a value to w_i. To remember the result of previous executions of the body, we use the boolean variable *beaten* in the program and the invariant will contain the predicate
$BEATEN(i) \equiv (beaten \leftrightarrow (\exists l < i : r_l > b_l))$.
Further, to determine T_{max}, we introduce time parameters T^i_{max}, for $i = 0, \ldots, k+1$ representing the maximum execution time up to and including the $i^{\rm th}$ iteration (the initialization if $i = 0$). Hence, we define

$$MOD_4 \equiv MOD_5 \; ; \; *[i \leq k \rightarrow MOD_6] \tag{M4}$$

with the intention that $[i \leq k \rightarrow MOD_6]$ should have the invariant

$$I \equiv time \leq t_0 + T^i_{max} \wedge \forall j < i : RW(j) \wedge BEATEN(i) \wedge i \leq k+1.$$

Program MOD_5 should initialize the variables to establish I, that is,

$$\langle\langle time = t_0 \rangle\rangle \; MOD_5 \; \langle\langle I \rangle\rangle.$$

Implementation of MOD_5

Observe that if $i = 0$ then $\forall j < i : RW(j) \wedge BEATEN(i)$ is equivalent to $BEATEN(0)$, that is, $beaten \leftrightarrow (\exists l < 0 : r_l > b_l)$. Since $\neg(\exists l < 0 : r_l > b_l)$, this is equivalent to $\neg beaten$. Hence it is easy to prove that the specification of MOD_5 is satisfied by

$$MOD_5 \equiv i := 0 \; ; \; beaten := false \tag{M5}$$

with $T^0_{max} = 2T_a$.

Implementation of MOD_6

To obtain formula (7) it remains to implement MOD_6 such that

$$\langle\langle I \rangle\rangle \; *[i \leq k \rightarrow MOD_6] \; \langle\langle time \leq t_0 + T_{max} \wedge \forall j \leq k : RW(j) \rangle\rangle. \tag{10}$$

As above, we could continue with the top-down derivation of program MOD_6 according to this specification. Here we directly give a program and show that it satisfies the specification. Let

$$\begin{array}{ll} MOD_6 \equiv & [\neg beaten \rightarrow w_i := b_i \; ; \; [r_i > b_i \rightarrow beaten := true] \\ & [\!]\, beaten \rightarrow w_i := 0 \\ &] \; ; \; i := i+1 \end{array} \tag{M6}$$

To prove (10) by the iteration rule, let $I_0 \equiv time \leq t_0 + T^i_{max}$ and observe that $I \rightarrow I_0$, $var(I_0) = \emptyset$, and $(\forall t_1 < \infty \, \exists t_1 > t_1 : t_2 \leq t_0 + T^i_{max}) \rightarrow false$. Hence we have to prove

$$\langle\langle I \wedge i \leq k \rangle\rangle \; [i \leq k \rightarrow MOD_6] \; \langle\langle I \rangle\rangle, \tag{11}$$

$$\langle\langle I \wedge i > k \rangle\rangle \; [i \leq k \rightarrow MOD_6] \; \langle\langle time \leq t_0 + T_{max} \wedge \forall j \leq k : RW(j) \rangle\rangle. \tag{12}$$

The proof of (11) is given by means of proof outlines. A proof outline is an *annotated program*, i.e., a program with assertions at control points, that satisfies certain verification conditions corresponding to the rules and axioms of the proof system.

Such a proof outline is often used as a concise representation of a correctness proof. First we consider the functional, non-real-time, part of the invariant. The proof of

$$\begin{array}{l}\langle\langle \forall j < i : RW(j) \wedge BEATEN(i) \wedge i \leq k+1 \wedge i \leq k \rangle\rangle \\ \quad [i \leq k \rightarrow MOD_6] \\ \langle\langle \forall j < i : RW(j) \wedge BEATEN(i) \wedge i \leq k+1 \rangle\rangle \end{array} \tag{13}$$

is given by the proof outline of table 3.

Table 3. Proof outline functional part.

$\langle\langle \forall j < i : RW(j) \wedge BEATEN(i) \wedge i \leq k+1 \wedge i \leq k \rangle\rangle$

$[\mathbf{i} \leq \mathbf{k} \rightarrow \langle\langle \forall j < i : RW(j) \wedge BEATEN(i) \wedge i \leq k \rangle\rangle$

$\quad [\neg\mathbf{beaten} \rightarrow \langle\langle \forall j < i : RW(j) \wedge \neg beaten \wedge \forall l < i : r_l \leq b_l \wedge i \leq k \rangle\rangle$

$\quad\quad \mathbf{w_i} := \mathbf{b_i}\,;$

$\quad\quad \langle\langle \forall j \leq i : RW(j) \wedge \neg beaten \wedge \forall l < i : r_l \leq b_l \wedge i \leq k \rangle\rangle$

$\quad\quad [\mathbf{r_i} > \mathbf{b_i} \rightarrow \langle\langle \forall j \leq i : RW(j) \wedge \exists l \leq i : r_l > b_l \wedge i \leq k \rangle\rangle$

$\quad\quad\quad \mathbf{beaten} := \mathbf{true}$

$\quad\quad\quad \langle\langle \forall j \leq i : RW(j) \wedge beaten \wedge \exists l < i+1 : r_l > b_l \wedge i \leq k \rangle\rangle$

$\quad\quad]\ \langle\langle \forall j \leq i : RW(j) \wedge BEATEN(i+1) \wedge i \leq k \rangle\rangle$

$\quad [\!]\,\mathbf{beaten} \rightarrow \langle\langle \forall j < i : RW(j) \wedge beaten \wedge \exists l < i : r_l > b_l \wedge i \leq k \rangle\rangle$

$\quad\quad \mathbf{w_i} := \mathbf{0}$

$\quad\quad \langle\langle \forall j \leq i : RW(j) \wedge BEATEN(i+1) \wedge i \leq k \rangle\rangle$

$\quad]\,;\ \langle\langle \forall j \leq i : RW(j) \wedge BEATEN(i+1) \wedge i \leq k \rangle\rangle$

$\quad \mathbf{i} := \mathbf{i} + \mathbf{1}$

$\quad \langle\langle \forall j < i : RW(j) \wedge BEATEN(i) \wedge i \leq k+1 \rangle\rangle$

$]\ \langle\langle \forall j < i : RW(j) \wedge BEATEN(i) \wedge i \leq k+1 \rangle\rangle$

To prove the timing part of the invariant, we could perform a worst case calculation, leading to the the requirement $T^i_{max} \geq T^{i-1}_{max} + 3T_g + 3T_a$. This upper bound is obtained from the branch which is executed if $\neg beaten$ and $r_i > b_i$. Observe, however, that this branch is executed at most once, and hence the bound on the maximum execution time can be decreased. Therefore we introduce a predicate $first(i)$ which expresses that i is the first position where the arbitration number of the module is beaten by the value of the bus line. Thus define
$first(i) \equiv r_i > b_i \wedge \forall l < i : r_l \leq b_l$.
Then, by the proof outline of table 4 we can prove

$$\begin{array}{l}\langle\langle time \leq t_0 + T^i_{max} \wedge\ BEATEN(i) \wedge i \leq k \rangle\rangle \\ \quad [i \leq k \rightarrow MOD_6] \\ \langle\langle\ (time \leq t_0 + T^{i-1}_{max} + 3T_g + 3T_a \wedge first(i-1)) \vee \\ \quad (time \leq t_0 + T^{i-1}_{max} + 3T_g + 2T_a \wedge \neg first(i-1)) \rangle\rangle. \end{array}$$

Table 4. Proof outline timing part.

$\langle\langle time \leq t_0 + T^i_{max} \wedge\ BEATEN(i) \wedge i \leq k \rangle\rangle$

$[\mathbf{i} \leq \mathbf{k} \rightarrow \langle\langle time \leq t_0 + T^i_{max} + T_g \wedge\ BEATEN(i) \rangle\rangle$

$\quad [\neg\mathbf{beaten} \rightarrow \langle\langle time \leq t_0 + T^i_{max} + 2T_g \wedge\ \forall l < i : r_l \leq b_l \rangle\rangle$

$\quad\quad \mathbf{w_i} := \mathbf{b_i}\ ;$

$\quad\quad \langle\langle time \leq t_0 + T^i_{max} + 2T_g + T_a \wedge\ \forall l < i : r_l \leq b_l \rangle\rangle$

$\quad\quad [\mathbf{r_i} > \mathbf{b_i} \rightarrow \langle\langle time \leq t_0 + T^i_{max} + 3T_g + T_a \wedge\ r_i > b_i \wedge \forall l < i : r_l \leq b_l \rangle\rangle$

$\quad\quad\quad \mathbf{beaten} := \mathbf{true}$

$\quad\quad\quad \langle\langle time \leq t_0 + T^i_{max} + 3T_g + 2T_a \wedge\ first(i) \rangle\rangle$

$\quad\quad]\ \langle\langle\ (time \leq t_0 + T^i_{max} + 3T_g + 2T_a \wedge first(i)) \vee$
$\quad\quad\quad (time \leq t_0 + T^i_{max} + 3T_g + T_a \wedge \neg first(i)) \rangle\rangle$

$\quad [\!]\ \mathbf{beaten} \rightarrow \langle\langle time \leq t_0 + T^i_{max} + 2T_g \wedge\ \exists l < i : r_l > b_l \rangle\rangle$

$\quad\quad \mathbf{w_i} := \mathbf{0}$

$\quad\quad \langle\langle time \leq t_0 + T^i_{max} + 2T_g + T_a \wedge\ \neg first(i) \rangle\rangle$

$\quad]\ ;\ \langle\langle\ (time \leq t_0 + T^i_{max} + 3T_g + 2T_a \wedge first(i)) \vee$
$\quad\quad (time \leq t_0 + T^i_{max} + 3T_g + T_a \wedge \neg first(i)) \rangle\rangle$

$\quad \mathbf{i} := \mathbf{i} + \mathbf{1}$
$\quad \langle\langle\ (time \leq t_0 + T^{i-1}_{max} + 3T_g + 3T_a \wedge first(i-1)) \vee$
$\quad\quad (time \leq t_0 + T^{i-1}_{max} + 3T_g + 2T_a \wedge \neg first(i-1)) \rangle\rangle$

$]\ \langle\langle\ (time \leq t_0 + T^{i-1}_{max} + 3T_g + 3T_a \wedge first(i-1)) \vee$
$\quad (time \leq t_0 + T^{i-1}_{max} + 3T_g + 2T_a \wedge \neg first(i-1)) \rangle\rangle$

Hence

$$\begin{array}{l} \langle\langle time \leq t_0 + T^i_{max} \wedge\ BEATEN(i) \wedge i \leq k \rangle\rangle \\ \quad [i \leq k \rightarrow MOD_6] \\ \langle\langle time \leq t_0 + T^i_{max} \rangle\rangle \end{array} \tag{14}$$

if, for $i = 1, \ldots, k+1$, $T^i_{max} = \begin{cases} T^{i-1}_{max} + 3T_g + 3T_a & \text{if } first(i-1) \\ T^{i-1}_{max} + 3T_g + 2T_a & \text{if } \neg first(i-1) \end{cases}$ (Def2)

Then by (13) and (14) we obtain (11). Further, to prove (12), we first show

$$\langle\langle I \wedge i > k \rangle\rangle \ \ \mathbf{delay}\ T_g \ \ \langle\langle time \leq t_0 + T^{k+1}_{max} + T_g \wedge \forall j \leq k : RW(j) \rangle\rangle. \tag{15}$$

This can be derived by the delay axiom, since $I[t_1/time] \wedge i > k \wedge time = t_1 + T_g$ implies $t_1 \leq t_0 + T^i_{max} \wedge \forall j < i : RW(j) \wedge i \leq k+1 \wedge i > k \wedge time = t_1 + T_g$, which leads to $time \leq t_0 + T^i_{max} + T_g \wedge \forall j < i : RW(j) \wedge i = k+1$, and thus commitment $time \leq t_0 + T^{k+1}_{max} + T_g \wedge \forall j \leq k : RW(j)$.

Next we compute T^{k+1}_{max} using (Def2). Therefore observe that there exists at most one value $i \in \{1, \ldots, k+1\}$ such that $first(i-1)$. Then, by (Def2), we obtain $T^{k+1}_{max} = T^0_{max} + 3(k+1)T_g + 2(k+1)T_a + T_a$. Using $T^0_{max} = 2T_a$ this implies

$T_{max}^{k+1} = 3(k+1)T_g + 2(k+1)T_a + 3T_a$.
Then by the guarded command rule (15) leads to (12), with

$$T_{max} = T_{max}^{k+1} + T_g = 3(k+1)T_g + 2(k+1)T_a + 3T_a + T_g. \qquad \text{(Def3)}$$

Final Implementation of MOD

If we combine (M), (M0), (M1), (M2), (M3), (M4), (M5) and (M6) we obtain the program of table 5. Note that the requirements (RM1) and (RM2) are fulfilled.

Table 5. Implementation of *MOD*.

```
x0 := clock ; apply(link, b0, ..., bk) ; x := clock ;
* [x − x0 ≤ D → read(r0, ..., rk) ; i := 0 ; beaten := false ;
                * [i ≤ k → [¬beaten → wi := bi ; [ri > bi → beaten := true]
                            [] beaten → wi := 0
                            ] ; i := i + 1
                  ] ;
                apply(link, w0, ..., wk) ; x := clock
  ]
```

Hence for this program we have

$\langle\langle p_0 \rangle\rangle$ MOD $\langle\langle C_0 \wedge C_{12345} \rangle\rangle$

provided the time constraints (T0) through (T7) are satisfied. Observe that (T7) implies (T0), (T2), and (T4). For (T3) note that, using (T7) and the definition of T, $\delta_0 + 2T_c + T_w + T_g \leq \Delta_0 + \Delta \leq MAX\Delta_0 + MAX\Delta \leq T \leq T + \delta$, i.e., (T3). By (T6), $\delta > T_s - T + T_{max} + T_c + 2T_g + T_r + T_w$. Thus δ is minimal if T_s is minimal. By (Def1), T_s is minimal if D is minimal, and hence by (T5), let

$D = \frac{(T - T_g - T_c)u}{g(1-r)} + u$.

Then we obtain a correct implementation provided

$\Delta_0 \geq \delta_0 + T_c$

$\delta > T_s - T + T_{max} + T_c + 2T_g + T_r + T_w$

$\Delta \geq T_c + T_g + T_r + T_w + T_{max}$

where T_s is defined by (Def1), that is,

$T_s \equiv \frac{(D+u)g(1+r)}{u} + T_c + \delta_0$

and T_{max} is defined by (Def3), i.e.,

$T_{max} = 3(k+1)T_g + 2(k+1)T_a + 3T_a + T_g$.

Finally, observe that with $\Delta = T_c + T_g + T_r + T_w + T_{max}$ for all modules, we can compute stabilization time T in terms of the bounds δ_0^i on the starting times of the modules, the parameter k denoting the number of lines of the bus, and a number of parameters representing the timing of the basic programming constructs.

7. Concluding Remarks

As in any theory for concurrency, the axiomatization of the communication mechanism of parallel processes is an important part of our real-time theory. In previous work [5, 7] we have formulated compositional proof systems based on Hoare triples for concurrent processes that communicate by synchronous or asynchronous message passing along channels. In this paper we have extended this formalism by axiomatizing the real-time properties of parallel modules that communicate via a common bus. The resulting proof method has been applied to an example of a distributed real-time arbitration protocol. In chapter 3 we have used compositionality to verify the protocol using only the specifications of the participating modules. By abstracting from implementation details we can concentrate on the essential properties of the algorithm.

When reasoning on the level of concurrent modules, as in chapter 3, we mainly use the commitments of our correctness formulae. Similar to classical Hoare triples, the structure of our assumption/commitment formulae is very suitable for the top-down design of a sequential programs. This has been demonstrated in chapter 6 where a real-time program has been derived satisfying the specification of a module as used in chapter 3. Observe that there are several correct implementations, and other design choices might lead to different programs. For instance, we could decide to split the innermost iteration into two parts where the first part applies the values of the arbitration number of the module until it is beaten, and the second part only applies 0.

An advantage of our assertional method is that we can easily deal with systems, such as $MOD_0 \| \ldots \| MOD_n$ here, that have a large number of possible states. Also observe that our generalization of the protocol, using general values instead of bits, does not complicate the specification and the verification. Furthermore, we can take parameters for the number of lines, the number of modules, and the execution times of atomic constructs, and derive relations between these parameters.

To evaluate our approach we intend to consider other examples of distributed real-time and fault-tolerant systems. It is especially interesting to investigate applications with different ways of communication between parallel processes. To axiomatize a new communication mechanism we have to adapt the assertion language with suitable primitives (e.g., to denote send and receive actions). Further, the proof system is extended with axioms to relate these primitives (such as the relation between send and receive actions). The proof system should also contain rules to relate communication statements of the programming language to the corresponding primitives of the assertion language. Rules and axioms which are not related to the communication mechanism remain the same.

Acknowledgements

I would like to thank Victor Yodaiken for suggesting and explaining the example of the real-time arbitration protocol to me.

References

[1] Zhou Chaochen, C.A.R. Hoare, and A.P. Ravn. A calculus of durations. *Information Processing Letters*, 40:269–276, 1991.

[2] de Bakker, Huizing, de Roever, and Rozenberg, editors. *Proceedings REX Workshop on Real-Time: Theory in Practice*. LNCS 600. Springer-Verlag, 1992.

[3] V.H. Haase. Real-time behaviour of programs. *IEEE Transactions on Software Engineering*, SE-7(5):494–501, 1981.

[4] E. Harel, O. Lichtenstein, and A. Pnueli. Explicit clock temporal logic. In *Proceedings Symposium on Logic in Computer Science*, pages 402–413. IEEE, 1990.

[5] J. Hooman. *Specification and Compositional Verification of Real-Time Systems*. LNCS 558, Springer-Verlag, 1991.

[6] J. Hooman. A compositional method for the top-down design of real-time systems. In *Proceedings 4th Euromicro Workshop on Real-Time Systems*, pages 86–91. IEEE, 1992.

[7] J. Hooman. A compositional approach to the design of hybrid systems. In *Workshop on Theory of Hybrid Systems*. LNCS, 1993.

[8] The Institute of Electrical and Electronics Engineers, Inc. *IEEE Standard Backplane Bus Specification for Multiprocessor Architectures: Futurebus*, 1988.

[9] R. Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.

[10] Z. Manna and A. Pnueli. Verification of concurrent programs: a temporal proof system. In *Foundations of Computer Science IV, Distributed Systems: Part 2*, volume 159 of *Mathematical Centre Tracts*, pages 163–255, 1982.

[11] J. Ostroff. *Temporal Logic for Real-Time Systems*. Advanced Software Development Series. Research Studies Press, 1989.

[12] F. Schneider, B. Bloom, and K. Marzullo. Putting time into proof outlines. In *Workshop on Real-Time: Theory in Practice*, pages 618–639. LNCS 600, Springer-Verlag, 1992.